Universidad Carlos III de Madrid

Escuela Politécnica Superior

Proyecto Fin de Carrera

Ingeniería en Informática

Goldfinch: Automatic Speech Recognition Platform

Autor: Daniel Ortega Ufano

Tutor: Luis Puente Rodríguez

Septiembre de 2012

Goldfinch: Automatic Speech Recognition Platform

Autor: Daniel Ortega Ufano

Director: Luis Antonio Puente Rodríguez

La defensa del presente Proyecto Fin de Carrera se realizó el día 24 de Septiembre de 2012 y fue evaluada por el siguiente tribunal:

Presidente: Mª Jesús Poza Lara

Secretario: Mónica Marrero Llinares

Vocal: José Luis López Cuadrado

Agradecimientos

En primer lugar, quiero dedicar este proyecto a mis padres por facilitarme el acceso a los estudios universitarios.

En segundo lugar, quiero agradecer a mi entonces novia y hoy mujer, María, la paciencia y comprensión que ha demostrado durante todo este tiempo de realización del proyecto, en el que apenas hemos podido disfrutar de nuestro primer año de convivencia juntos. Ella ha sido una pieza fundamental sin la que no hubiera conseguido finalizar mis estudios.

Este agradecimiento es también para mis abuelos, que durante la carrera han aguardado ilusionados las notas de cada una de las convocatorias de exámenes y que aún hoy están expectantes por la finalización de mis estudios.

A mis hermanos, que espero comprendan que con mayores motivaciones y esfuerzo, es posible la consecución de sus estudios. Todo será gracias a vuestro tesón y dedicación.

A mis compañeros de la universidad por los intensos momentos vividos durante todos estos años, y con los cuales no sólo he compartido prácticas y lugar de estudio.

También quiero agradecerles a mis compañeros de idealista.com (el cual constituye mi actual lugar de trabajo) todo el conocimiento que me han aportado, y que en cierto modo han contribuido a la realización del proyecto en su parte técnica.

Quiero agradecer también a mi tutor la ayuda y apoyo demostrado a lo largo de todo el proyecto, facilitando las reuniones y comprendiendo mi situación personal para establecer los plazos de entrega. Sin su aporte hubiera sido mucho más complicada la realización de este trabajo.

Por último, quiero agradecer a la comunidad de desarrollo del software libre, porque muchas de sus herramientas han sido utilizadas satisfactoriamente en este proyecto.

Y aunque como dice el dicho "son todos los que están, pero no están todos los que son", lo único que puedo decir es: **Muchísimas gracias de todo corazón.**

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

La tecnología del reconocimiento automático del habla (en adelante ASR, siglas del término inglés Automatic Speech Recognition) ya suscitaba un creciente interés en el siglo XIX antes de la creación y uso generalizado de las computadoras, las cuales actualmente son la principal plataforma de ejecución de esta tecnología.

A finales del siglo XIX Tihamer Nemes solicitó una patente para el desarrollo de una máquina que transcribiera automáticamente voz a texto escrito en papel [Carpenter, et al. 1991], pero el proyecto fue rechazado por ser considerado "poco realista". Ya entonces se vislumbraba la dificultad que podría entrañar el desarrollo de una tecnología que lo permitiera. Treinta años después AT&T Bell Laboratories creó la primera máquina capaz de reconocer voz, cuya única capacidad era transcribir automáticamente de voz a texto escrito los 10 dígitos de inglés. Requería un extenso reajuste y entrenamiento a la voz de una persona, pero era extremadamente fiable pues tenía una tasa del 99% de acierto.

Un siglo más tarde, a mediados de los años 60, los investigadores admitieron que el reconocimiento del habla era un proceso más intrincado y sutil de lo que habían anticipado. No fue hasta la década de los setenta cuando nació el primer producto de reconocimiento de voz, el VIP100 de Threshold Technology Inc., que contaba con un vocabulario poco extenso, era dependiente del locutor y reconocía palabras de forma aislada [Hierro, 2004].

El interés por la inteligencia artificial surgido en esta época en la agencia de proyectos de investigación ARPA propició un fuerte desarrollo de esta tecnología. También contribuyó el aumento de la capacidad de cómputo y almacenamiento de los ordenadores y otros dispositivos a finales del siglo XX. Todo ello permitió la implementación de algoritmos de ASR con excelentes resultados en computadores, reconociendo cada vez vocabularios más amplios.

Hoy en día existen numerosos dispositivos que utilizan la tecnología de ASR. Desde teléfonos móviles (como Apple con su reciente sistema de reconocimiento denominado "Siri") o navegadores GPS en los que se pueden programar itinerarios sin tener que hacer uso de las manos.

En la actualidad estos dispositivos se están popularizando como núcleo de los sistemas de subtitulación automática. Dicha subtitulación facilita el acceso de las personas con deficiencias auditivas a la información multimedia [Revuelta, et al. 2008] en cine, televisión e internet.

No cabe duda que los distintos logros en el campo de ASR facilitan la vida de las personas y que nuevas investigaciones permitirán la aplicación de esta disciplina en otras áreas. Se espera que nuevos avances permitan una comunicación más humana y natural con todo tipo de dispositivos electrónicos.

En las siguientes secciones del capítulo introductorio se ofrece una presentación del proyecto que será desarrollado a lo largo de esta memoria. Por una parte, se da a conocer la motivación que ha conducido al desarrollo de la Plataforma Goldfinch para el reconocimiento del habla y por otro los objetivos perseguidos.

1.1 Motivación

El reconocimiento automático del habla es un campo de investigación de creciente relevancia. El desarrollo de algoritmos más potentes y de modelado más preciso, junto con la aparición de sistemas informáticos más potentes y asequibles permiten su integración en numerosos ámbitos de la sociedad actual [Fandiño, 2005].

La comunidad científica y diferentes organizaciones comerciales (como por ejemplo *Nuance*® con su producto *Dragon Naturally Speaking*) ofrecen distintas plataformas y herramientas para el Reconocimiento Automático del Habla; sin embargo la mayoría de ellas no constituyen plataformas para la investigación y el aprendizaje de esta disciplina.

Se considera que el reconocimiento de voz automatizado es una tarea muy compleja debido a sus requerimientos implícitos. El problema que se plantea en un sistema de ASR es establecer mecanismos que coordinen un conjunto de informaciones provenientes de diversas fuentes de conocimiento (acústica, fonética, fonológica, léxica, y semántica) en presencia de ambigüedades, incertidumbres y errores inevitables para llegar a obtener una interpretación aceptable del mensaje acústico recibido [Casacuberta, 1987].

Dentro del departamento de Informática de la Universidad Carlos III de Madrid, se planteó el desarrollo de una plataforma de experimentación para la evaluación independiente de los distintos módulos que definen la cadena de procesos que se ejecutan en el proceso de reconocimiento del habla continua y el locutor.

El proyecto desde el comienzo ofrecía grandes posibilidades, puesto que la solución puede ser abordada de muchas formas diferentes.

1.2 Objetivos

El proyecto de la Plataforma Goldfinch nace a como consecuencia de la situación presentada en la sección anterior. El objetivo principal del proyecto es crear un entorno de pruebas, en el que se puedan evaluar y ajustar los módulos que combinados funcionen como un sistema de reconocimiento automático del habla.

El planteamiento y desarrollo de la plataforma implica la creación de un entorno estable y controlado que permita a los investigadores obtener resultados, evaluarlos y compararlos. Dichos resultados provienen de los módulos que implementan los distintos algoritmos y técnicas existentes para el reconocimiento del habla y el locutor.

El conjunto de los objetivos planteados para el proyecto es el siguiente:

- Adquirir el conocimiento suficiente sobre el estado del arte en el reconocimiento automático del habla que permita el desarrollo de una plataforma de reconocimiento del habla continua.

- Evaluar el estado de la técnica de los principales reconocedores existentes hasta la fecha.

- Desarrollar una plataforma que permita alojar a los distintos módulos que componen un reconocedor del locutor y del habla, y que sea capaz de gestionar los flujos de información entre estos.

- Diseñar una arquitectura multimódulo y definir una metodología de diseño para determinar la adecuación de los distintos módulos de cara a su uso en conjunto.

- Demostrar el correcto funcionamiento de la plataforma desarrollada a través de un conjunto de módulos de prueba que simulen el funcionamiento de los distintos procesos de los que se compone un reconocedor del habla.

El proyecto no se centra en el desarrollo e investigación de los módulos que componen el reconocedor del locutor y del habla. No obstante se deberán proporcionar unos módulos de prueba que verifiquen el correcto funcionamiento de la plataforma.

De forma más general con la realización del proyecto se pretende poner en práctica los distintos conocimientos adquiridos a lo largo de los estudios de Ingeniería Informática.

1.3 Estructura del documento

La memoria del proyecto recoge en las siguientes secciones la evolución del proyecto de desarrollo de la plataforma Goldfinch. La memoria se estructura en 9 capítulos, cuyo contenido se describe a continuación:

La introducción del proyecto, que refleja tanto las motivaciones como los objetivos que se pretenden alcanzar, se incluye en el primer capítulo.

En el segundo capítulo se define la articulación y percepción del habla, y la estructura del lenguaje hablado, introduciendo conceptos que serán utilizados a lo largo de todo el documento.

Posteriormente y antes de proceder a describir la propuesta realizada en el marco de este proyecto, en el tercer capítulo se repasa el estado de la técnica, detallando el funcionamiento de un ASR básico y revisando los principales sistemas de reconocimiento del habla existentes hasta la fecha.

En el cuarto capítulo se describe la arquitectura funcional de la plataforma donde se exponen detalladamente los diferentes componentes arquitectónicos de la plataforma. Dicho estudio se acompaña de la presentación de los casos de uso y de una descripción jerarquizada del diagrama de flujo de datos. Además se incluyen las capacidades y restricciones generales del sistema, y el catálogo de requisitos de software.

Como complemento al capítulo anterior, el quinto capítulo contiene una justificación de las decisiones tecnológicas adoptadas, incluyendo en este punto los Frameworks, lenguajes y herramientas utilizadas. El diseño de la plataforma propuesta se recoge en el sexto capítulo, donde se explican los patrones arquitectónicos y de diseño implementados. Además se recoge una sección para el diseño detallado.

Una vez acabada la parte técnica, el séptimo capítulo trata la gestión del proyecto, donde se explica el plan de ejecución y se incluye una estimación de recursos tanto temporales como económicos.

Por último en el octavo capítulo se presentan las posibles líneas de trabajo futuro y las conclusiones alcanzadas a lo largo de este proyecto. Adicionalmente las referencias bibliográficas mencionadas a lo largo del documento se presentan en el noveno capítulo.

Se incluyen cinco apéndices: En el primero se recoge toda la terminología empleada dividida en vocabulario y acrónimos. El segundo adjunta los requisitos de usuario y de software y una matriz de trazabilidad a modo de resumen que los relaciona. El tercero se refiere al catálogo de casos de uso. El cuarto incluye el prototipo de la interfaz gráfica con su correspondiente descripción y el último recoge una breve reseña de los módulos de prueba desarrollados.

2. Articulación del habla y estructura del lenguaje hablado

En este capítulo se ofrece una introducción sobre las características más importantes del sonido. Posteriormente se detalla la articulación del habla y los órganos fonoarticulatorios que participan en ella. Después se describe brevemente la percepción del habla y por último se define la estructura del lenguaje hablado.

2.1 Articulación del habla

Las señales del habla están compuestas por patrones de sonidos analógicos, que sirven como base para una representación discreta y simbólica de un idioma hablado. Dicha representación incluye a los fonemas, las sílabas y las palabras. La producción e interpretación de estos sonidos se rigen por la sintaxis y semántica del idioma hablado.

En esta sección se incluye una definición acerca del sonido, se detallan pormenorizadamente sus características y se describen los órganos fonoarticulatorios que participan en la articulación del habla.

2.1.1 Sonido

El sonido es una onda de presión longitudinal formada por compresiones y expansiones del aire, en dirección paralela a la aplicación de energía. Las compresiones son áreas donde las moléculas de aire han sido forzadas por la aplicación de energía, dando lugar a una mayor concentración de las mismas. Las expansiones sin embargo son áreas donde la concentración de aire es menor [Huang, et al. 2001].

Aunque típicamente se identifica al aire como medio de propagación de la onda, el sonido se puede propagar también a través de un fluido (u otro medio elástico). En los fluidos el sonido se propaga a través de fluctuaciones de presión. En cuerpos sólidos elásticos involucra variaciones del estado tensional del medio. A lo largo del documento se supondrá el aire como medio de propagación del sonido.

Comúnmente las configuraciones alternativas de compresión y rarefacción de las moléculas de aire desde la fuente de energía se representan como una onda sinusoidal.

Dicha onda sinusoidal se muestra en la siguiente figura[1]:

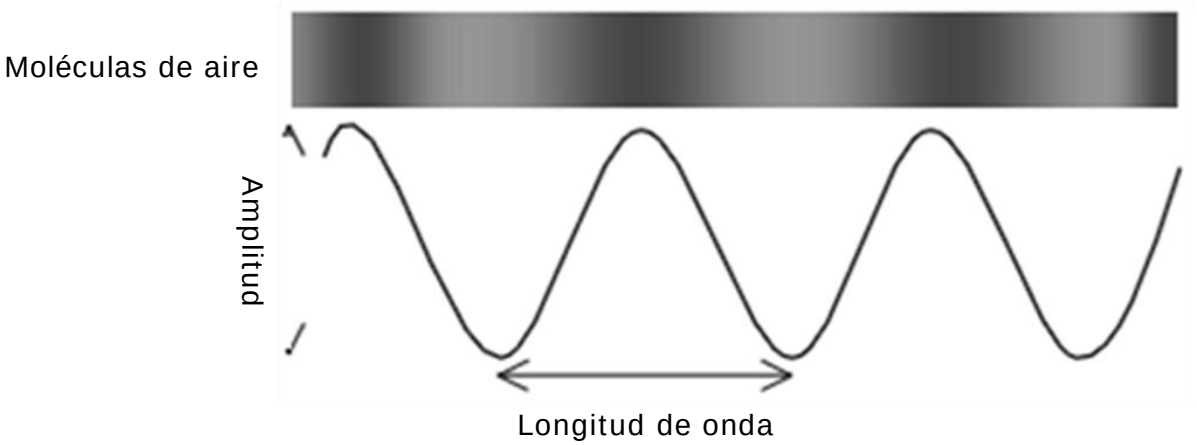


Figura 1 - Onda de presión del sonido

Características del sonido

Amplitud

La amplitud de una onda de sonido es el grado de movimiento de las moléculas de aire en una onda de presión. Se corresponde a la intensidad de la rarefacción y compresión que la acompañan.

Puede expresarse en unidades absolutas midiendo la distancia del desplazamiento de las moléculas del aire, o también midiendo la diferencia de presiones entre la compresión y la rarefacción. Habitualmente la amplitud del sonido se mide utilizando una escala logarítmica que utiliza como unidad los decibelios (dB). La escala de decibelios es en realidad una manera de comparar dos sonidos, y se puede expresar del siguiente modo:

$$10\log_{10}(P1/P2)$$

Donde $P1$ y $P2$ son los dos niveles de potencia.

Frecuencia

La frecuencia de un sonido puede percibirse como tonos más graves o más agudos. La frecuencia es el número de oscilaciones que una onda sonora efectúa en un tiempo dado y se mide en hertzios. El rango de frecuencia audible por un ser humano se sitúa entre los 20 y los 20.000 hertzios aproximadamente.

[1] Fuente: [Huang, et al. 2001]

Longitud de onda

La longitud de una onda refleja el periodo espacial o la distancia que hay entre dos puntos consecutivos que poseen la misma fase. La longitud de onda se relaciona con la frecuencia de la siguiente manera:

$$\lambda = \frac{v}{f} \quad \text{(m)}$$

Siendo:

λ : Longitud de onda

v : Velocidad de propagación de la onda

f : Frecuencia de la onda

Velocidad

La velocidad aproximada de una onda de presión de sonido en el aire viene descrita por la siguiente fórmula:

$$v = 331{,}5 + 0{,}6T_c \quad \text{(m/s)}$$

Siendo:

T_c : Temperatura en grados Celsius

Nivel de presión sonora

El nivel de presión del sonido (SPL) es una medida de la presión absoluta del sonido P en dB:

$$SPL = 20\log_{10}\left(\frac{P}{P_0}\right) \quad \text{(dB)}$$

Siendo:

P : Presión sonora instantánea

P_0 : Valor para la presión de referencia de referencia = 0,0002 μ bar para un tono de 1kHz.

Presión acústica

La energía provocada por las ondas sonoras genera un movimiento ondulatorio de las partículas del aire, provocando la variación alterna en la presión estática del aire. Como ya se describió con anterioridad se producen áreas donde se concentran partículas de aire (zonas de compresión o concentración) y áreas donde hay menor concentración de partículas de aire (zonas de rarefacción).

Las zonas de compresión tienen mayor densidad que las de rarefacción por la presencia de un mayor número de partículas de aire.

Al aplicar fuerza a las partículas de aire, estas se comprimen y expanden alternativamente, lo que queda reflejado en las pequeñas variaciones de la presión atmosférica. La presión acústica se mide en Pascales, y la mínima presión acústica que puede percibir el oído humano es de $2x10^{-5}$ Pascales.

Intensidad Sonora

La intensidad del sonido es el flujo medio de energía por unidad de área perpendicular a la dirección de propagación. En el caso de ondas esféricas que se propagan desde una fuente puntual, el valor que toma la intensidad es inversamente proporcional al cuadrado de la distancia (conocida como la ley del inverso al cuadrado). En este caso se supone que no se producen pérdidas de energía debido a la viscosidad, la conducción térmica u otros efectos de absorción. En la propagación real de un sonido en la atmósfera, este se ve afectado por cambios de las propiedades físicas del aire como la temperatura, presión o humedad, y se produce una amortiguación y dispersión de sus ondas.

La intensidad se mide en Watios/m^2.

El valor de la intensidad depende del campo acústico donde se encuentre el sonido:

- Para una onda acústica esférica progresiva

  $$I = \frac{p^2}{\rho c} \ (W/m^2)$$

- En un campo difuso cerca de una pared

  $$I = \frac{p^2}{4\rho c} \ (W/m^2)$$

Siendo:

p : Presión sonora que crea la onda

ρ : Densidad del medio donde se propaga la onda

c : Velocidad del sonido en el medio

2.1.2 Voz

La voz es el sonido que se produce cuando el aire espirado vibra al pasar por la laringe y por las cavidades situadas en su parte superior: faringe, fosas nasales y boca.

En el habla intervienen los siguientes órganos fonoarticulatorios y resonadores [Rodríguez, et al. 2004]:

- Laringe
- Faringe
- Fosas nasales
- Fauces
- Boca

Laringe

La laringe es parte del sistema respiratorio y constituye el principal órgano fónico. Recibe el aire espirado y lo hace vibrar a su paso. Para ello posee un complejo sistema muscular revestido de mucosa cuyos principales componentes son los pliegues vocales que vibran para producir la voz con sus diferentes cualidades de intensidad, altura tonal y timbre.

Faringe

La faringe está situada entre el esófago, la laringe, las fosas nasales y la boca. Por su localización, participa en las funciones digestivo-respiratorias. Recibe la columna de aire fónico que sale de la laringe. Como sus paredes están formadas por unos músculos constrictores, tiene capacidad para cambiar de forma, posición y volumen, lo que le permite intervenir en la resonancia y articulación de la voz.

Fosas nasales

Las fosas nasales son dos cavidades que se comunican con el exterior por medio de orificios nasales. Por detrás están comunicadas con la faringe. Están situadas superiormente a la boca y, aparte de servir para la olfacción y la limpieza y conducción de aire hacia los pulmones, intervienen en la resonancia y timbre, pero no tienen capacidad articulatoria.

Fauces

Las fauces corresponden a la zona de paso entre faringe y boca. Se encuentran delimitadas por las siguientes estructuras musculares:

- Paladar blando
- Músculos de los pilares del istmo de las fauces
- Porción posterior de la lengua

Su actividad muscular es importante para articular sonidos guturales.

Cavidad bucal

La cavidad bucal actúa como un resonador. Posee importantes estructuras para la articulación del habla. El aire espirado después de pasar por la laringe, faringe y las fauces, llega a la boca donde es sometido a vibraciones, a interrupciones y a escapes intermitentes, convirtiendo los sonidos en algo que tenga significado fonético. Las estructuras de la boca que intervienen en la articulación son las siguientes: lengua, paladar, mandíbula y los músculos que la movilizan, los dientes, los labios y otros músculos faciales.

Sus funciones aparte de la articulación del lenguaje son la función respiratoria, masticatoria y deglutoria. Desde el punto de vista biológico estas últimas son más vitales e importantes que la propia función del habla, que fue adquirida más tardíamente.

2.2 Percepción del Habla

En el sistema de percepción del habla está compuesto por los órganos auditivos periféricos (oídos) y el sistema nervioso auditivo (cerebro).

El oído procesa señales de presión acústica del siguiente modo: Primero las transforma en un patrón de vibraciones mecánicas sobre la membrana basilar y posteriormente representa los patrones como una serie de pulsos transmitidos por el nervio auditivo.

El oído humano se descompone en tres partes:

- **Oído externo:** Consiste en la parte exterior visible y el canal auditivo externo que forma un tubo por donde circula el sonido. Tiene aproximadamente 2,5 cm de longitud.

- **Oído medio:** Su función es transformar la energía acústica en energía mecánica para posteriormente transmitirla al oído interno (a este proceso se le conoce como transducción).

- **Oído interno:** Transforma la energía mecánica en impulsos eléctricos. La estructura más relevante del oído interno es la cóclea, la cual está comunicada directamente con el nervio auditivo, que es el encargado de conducir la representación del sonido al cerebro. La cóclea forma una espiral de aproximadamente 3,5 cm de longitud.

2.2.1 Análisis espectral

Como se ha descrito anteriormente, la cóclea es una espiral que se comunica con el nervio auditivo. El funcionamiento de dicha espiral se asemeja al de un banco de filtros.

AT&T Bell Labs desde comienzos del siglo XX ha contribuido de manera muy influyente en distintos descubrimientos sobre audición, centrándose en dos campos: *bandas críticas* e *índices de articulación*.

Una banda crítica se define como el intervalo de frecuencias que representa la máxima resolución en frecuencia del sistema auditivo obtenido por diferentes experimentos psicoacústicos [Suárez, 2007].

Existen otros trabajos que apuntan a la existencia de bandas críticas en la respuesta de la cóclea. Dichas bandas son de suma importancia para entender la percepción de la intensidad, el tono y el timbre.

El sistema de percepción del habla lleva a cabo un análisis espectral del sonido. La cóclea actúa como si estuviera compuesta por filtros superpuestos cuyo ancho de banda es el mismo que el ancho de banda crítico.

Existe una escala de bandas críticas denominada Escala de frecuencias de Bark. La escala tiene un rango de 1 a 24 *Barks*, correspondientes a las 24 bandas críticas de la audición. La resolución perceptiva es mejor para las bajas frecuencias. Nótese que las bandas críticas del oído son continuas, y un tono de cualquier frecuencia audible siempre encuentra una banda crítica centrado en dicha frecuencia.

La frecuencia de Bark puede ser expresada como:

$$b(f) = 13 * \arctan(0{,}00076 * f) + 3{,}5 * \arctan((f/7500)^2) \quad \text{(Bark)}$$

Otra escala ampliamente utilizada es la escala de Mel que puede aproximarse como:

$$b(f) = 1125 * \ln(1 + f/700) \quad \text{(mel)}$$

Esta escala se utiliza ampliamente en los reconocedores del habla más modernos [Huang, et al. 2001].

2.3 Estructura del lenguaje hablado

Se entiende como lenguaje a la capacidad innata que poseen los seres humanos para comunicarse, el cual puede ser verbal o no verbal. En este estudio únicamente se abordará el lenguaje verbal.

El sistema de la lengua, dada su complejidad, se estructura y descompone a su vez en otros sistemas o niveles lingüísticos, relacionados entre sí, y que cubren distintos aspectos del lenguaje. Estos subsistemas son los siguientes: fónico, morfológico, sintáctico y semántico.

El nivel fónico se ocupa del plano oral de la lengua, cuyos elementos constitutivos son el fonema y el sonido. El fonema es la unidad mínima abstracta, que no posee significado, pero en cambio si presenta una serie de rasgos que le confieren un valor distintivo y sirven a su vez para diferenciar palabras. El apartado "Fonética y Fonología" contiene una definición más amplia sobre estos conceptos.

El nivel morfosintáctico se centra en la lengua desde el punto de vista morfológico y sintáctico. Mientras que el nivel morfológico estudia las palabras, los morfemas y su estructura, el nivel sintáctico estudia la función de las palabras en un contexto determinado. Aunque se suelen tratan como un solo nivel se presentan en dos apartados; El apartado "Sílabas y palabras" trata sobre el nivel morfológico. El nivel Sintáctico se aborda en el apartado "Sintaxis y semántica" junto con el nivel semántico que estudia el significado léxico de las palabras.

2.3.1 Fonética y Fonología

La fonética se refiere al estudio de los sonidos hablados y su producción, clasificación y transcripción. Fonología es el estudio de la distribución de los sonidos hablados en un lenguaje y las reglas tácitas de pronunciación.

Los fonemas son unidades teóricas, postuladas para estudiar el nivel fonético-fonológico de una lengua humana. Se requiere que exista una función distintiva para determinar que constituye o no un fonema: son sonidos del habla que permiten diferenciar palabras en una lengua. Por ejemplo, los sonidos */p/* y */b/* son fonemas del español porque existen palabras como */parca/* y */barca/* que tienen significado distinto y su pronunciación sólo difiere en esos dos sonidos.

Las palabras que tienen significados distintos y difieren únicamente en un sonido se denominan pares mínimos.

El número de fonemas de una lengua está limitado al número de alófonos potencialmente definibles, aunque si se definen rasgos fonéticos muy sutiles, es potencialmente ilimitado y varía según el contexto fonético y la articulación individual de los hablantes. La mayoría de los análisis del español utilizan 24 fonemas (5 vocales y 19 consonantes), aunque no todas las variedades de este idioma tienen el mismo número de fonemas.

La fonología por su parte no trata necesariamente entes distinguibles en términos acústicos. Desde el punto de vista estructural, mientras que un fonema pertenece a la lengua, el sonido asociado pertenece al habla. Un fonema por tanto no es un sonido con entidad física, es una abstracción mental o formal de los sonidos del habla.

El sistema fonológico de una lengua está formado por un inventario de fonemas y un conjunto de reglas de aplicación automática que proporciona la pronunciación de cada cadena admisible de fonemas. Un sistema fonológico se puede representar por el siguiente par:

Sist. Fonológico = $(\Re, F)$

Siendo:

$\Re$: Conjunto de reglas que permiten derivar la pronunciación fonética de una palabra a partir de la forma fonémica de una expresión.

F: Conjunto de fonemas o inventario fonológico.

Vocales

Como se ha descrito anteriormente en la articulación de las vocales no se producen constricciones ni obstrucciones del aire en la cavidad oral. La variación de la posición de la lengua es la que produce las distintas vocales, pues se efectúan cambios de la resonancia en la cavidad oral.

Para la producción de una vocal, el tracto vocal se comporta como un tubo cerrado en un extremo: la laringe, y abierto en otro: los labios. Las resonancias producidas en este tracto se denominan formantes, que se muestran en el espectro del sonido como los picos de mayor amplitud a una frecuencia determinada.

Cada vocal está determinada por sus tres primeros formantes, generados por la cavidad faríngea (primer formante o F1), la bucal (segundo formante o F2) y los labios (tercer formante o F3) [Martínez, 1998].

Un formante es un pico de intensidad en el espectro de un sonido. Puede definirse también como una concentración de energía que se refleja en una determinada frecuencia. En el habla vienen determinados por el proceso de filtrado por resonancia realizado en la cavidad faríngea.

Las vocales [i] y [u] poseen el F1 más bajo (alrededor de los 250 Hz), pues en ambas la masa de lengua se eleva, hacia el paladar en el caso de [i] o hacia el velo en [u]. Esa elevación supone un ensanchamiento de la faringe, que provoca que la frecuencia del F1 sea más grave. En el caso de la pronunciación de [a], la lengua se aplana y baja, y produce un estrechamiento de la faringe. En ese caso el F1 se aproxima a los 700 Hz. Las demás vocales, al ser intermedias, tendrán también valores situados entre esos dos extremos.

El F2 depende de la posición antero-posterior de la lengua en la boca. Cuanto más anterior más alto será (en el caso de [i] alrededor de los 2.200 Hz, mientras que en [u] la F2 desciende hasta los 700 Hz debido a que es la vocal más posterior).

El F3 ejerce una función importante en las lenguas que distinguen entre vocales anteriores y posteriores redondeadas y no redondeadas. Esto no sucede en el español pues las anteriores son no redondeadas y las posteriores son redondeadas sistemáticamente. Por este motivo el F3 tiene un papel menos importante en la identificación de la vocal.

En las gráficas mostradas en la figura 2 se pueden observar los formantes en el espectro y en el espectrograma.

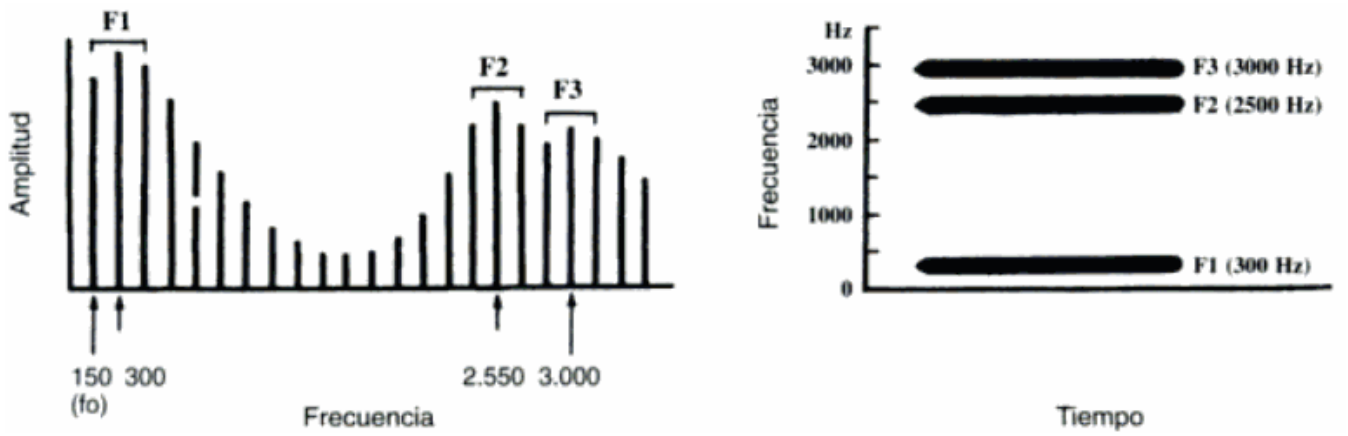


Figura 2 - Los formantes en el espectro y en el espectrograma

Consonantes

Al contrario que las vocales, las consonantes se caracterizan por las constricciones y obstrucciones que se producen en la faringe y/o la cavidad oral.

Para definir una consonante debe tenerse en cuenta el punto de articulación, el modo de articulación, la acción de las cuerdas vocales y la posición del velo del paladar [Sánchez, 1995].

El punto de articulación es el lugar donde se tocan o aproximan los órganos para producir el sonido. Por el punto de articulación pueden ser:

- **Bilabiales:** Se acercan o cierran los labios.
- **Labiodentales:** Se acerca el labio inferior a los incisivos superiores.
- **Interdentales:** Se aproxima la punta de la lengua a los dientes incisivos.
- **Dentales:** Se coloca el ápice de la lengua contra la pared interior de los incisivos superiores.
- **Alveolares:** Se acerca la punta de la lengua a los alvéolos.
- **Palatales:** Se levanta el dorso de la lengua contra el paladar.
- **Velares:** El dorso de la lengua toca o se acerca al velo del paladar.
- **Bilabiovelares:** Los labios contactan al tiempo que se pronuncia la consonante como si fuera velar.

Por el modo de articulación son:

- **Oclusivas:** Se produce una oclusión (bloqueo) completa de las cavidades oral y nasal del tracto vocal.
- **Fricativas:** Se produce una fricción continua en el punto de articulación.

- **Africadas:** Consonante que comienza como oclusiva pero al soltar el aire se convierte en fricativa.
- **Vibrantes:** Se produce mediante una oclusión momentánea de la cavidad oral.
- **Semiconsonantes:** Se pronuncia como una vocal, pero con la lengua más próxima al paladar, de modo que se origina una ligera turbulencia.

Por la acción de las cuerdas vocales son (en el apartado *Sonidos sordos y sonidos sonoros* se describe este punto con mayor detalle):

- Sonoras (con vibración)
- Sordas (sin vibración)

Por la posición del velo del paladar son:

- Orales
- Nasales
- Oronasales

Sonidos sordos y sonidos sonoros

Los tipos de sonidos se diferencian en si son sordos o sonoros. En los sonidos sonoros (incluyendo las vocales) se puede identificar tanto en tiempo como en frecuencia un patrón más o menos regular del cual carecen consonantes como la *s*. Normalmente los sonidos sonoros poseen más energía que los sordos.

La siguiente figura[2] muestra la forma de onda de la palabra inglesa *"sees"*, la cual está formada por tres fonemas: la consonante sorda */s/*, la vocal */iy/* y la consonante sonora */z/*.

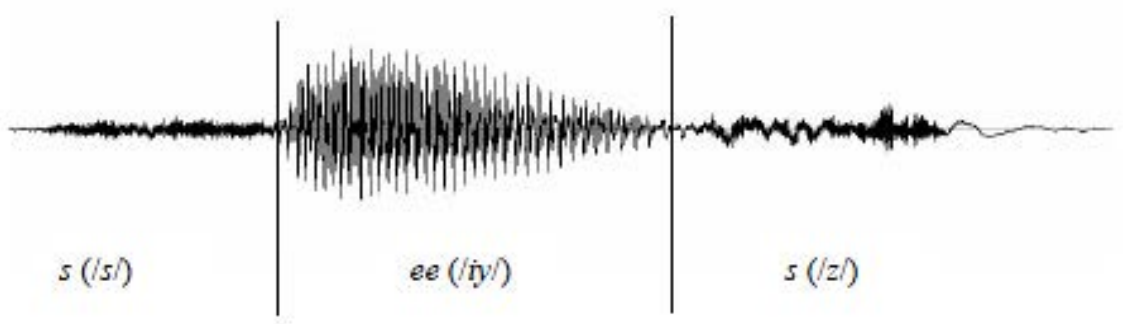


Figura 3 - Forma de onda de la palabra "sees"

[2] Fuente: [Huang, et al. 2001]

Esta distinción se realiza en base a la vibración de las cuerdas vocales; Si las cuerdas vocales vibran durante la articulación de un fonema, el fonema es considerado sonoro y si no lo hacen es considerado sordo.

2.3.2 Sílabas y palabras

Las sílabas y las palabras se encuentran en un orden superior a los fonemas. Para contribuir al significado del lenguaje, estas se deben organizar en unidades más cohesivas.

Sílabas

La sílaba es una unidad intermedia que se interpone entre los niveles de los fonemas y las palabras. En la mayoría de lenguas, las palabras pueden dividirse en bisílabas que constan de un núcleo silábico, un ataque silábico y una coda:

- El núcleo silábico es la parte central de la sílaba, que tiene mayor intensidad sonora, y que se manifiesta en el espectrograma con una mayor amplitud.
- El ataque silábico (también conocido como arranque silábico) es la parte de una sílaba que precede al núcleo de la misma.
- La coda es la consonante o el grupo consonántico en posición postnuclear dentro de una sílaba (después de la vocal nuclear).

En el castellano, el núcleo silábico está formado siempre por una vocal o diptongo (creciente o decreciente). El ataque silábico puede existir o no y como máximo puede estar formado por dos fonemas. La coda puede ser simple o doble.

Palabras

Se puede definir la palabra como la unidad mínima con significado que se puede pronunciar de manera aislada [Hualde, et al. 2002]. No obstante existen otras perspectivas para el estudio de la palabra:

- **Criterio fonológico:** La palabra es un segmento ligado por junturas, pausas o que constituyen el núcleo posible de un grupo acentual.
- **Criterio formal o fonológico:** La palabra es la mínima forma libre, caracterizada por la posibilidad de aparecer en cualquier posición de la cadena hablada.

- **Criterio funcional:** La palabra supone una unidad dotada de una función.
- **Criterio semántico:** Asociación de un sentido dado y un conjunto de sonidos dado dentro de una función gramatical.

El estudio de la estructura de las palabras requiere la utilización de un concepto más básico: el morfema. El morfema puede definirse como la unidad mínima con significado de la palabra. Una palabra puede estar formada por un único morfema, o contener afijos (en forma de prefijos que van delante de la raíz o sufijos que van después de la raíz) que contribuyen al significado de la palabra completa.

2.3.3 Sintaxis y semántica

La sintaxis es el estudio de los patrones de formación de frases a partir de palabras y las reglas para la formación de sentencias gramaticales [Huang, et al. 2001].

Dentro de las oraciones las palabras forman agrupaciones jerárquicas que se denominan constituyentes sintácticos. Dichos constituyentes realizan una función sintáctica reconocible y pueden generalmente descomponerse en dos subsecuencias o más. Cada una de estas subsecuencias es a su vez otro constituyente. El conjunto de todos los constituyentes de una oración es un conjunto parcialmente ordenado, en donde el orden se basa en la descomponibilidad de los constituyentes en subconstituyentes.

Las relaciones jerárquicas entre los constituyentes sintácticos pueden representarse mediante un árbol sintáctico, que es un grafo que se representa las relaciones de orden parcial.

Una secuencia es un constituyente si cumple alguna de las pruebas de constituencia:

- **Substitución pronominal:** Se puede sustituir la secuencia por un pronombre sin que se altere el significado de la frase.
- **Substitución convencional:** Se puede sustituir la secuencia por otra de tipo similar sin que cambie la jerarquía sintáctica.
- **Desplazamiento:** Se puede desplazar la secuencia al inicio sin alterar el significado.
- **Aislabilidad:** La secuencia es una respuesta válida a una pregunta.

Actualmente no existe un procedimiento algorítmico que permita establecer la estructura de los constituyentes de cualquier oración.

La semántica por su parte se refiere al estudio del sentido de las palabras, las frases y los enunciados [Tamba, 2005]. La semántica puede estudiarse desde distintos puntos de vista:

- Semántica lingüística: Trata de la codificación y decodificación de los contenidos semánticos en las estructuras lingüísticas.
- Semántica lógica: Estudia la relación entre el signo lingüístico y la realidad.
- Semántica en ciencias cognitivas: Intenta explicar el sentido de la comunicación y el mecanismo psíquico que se establece entre hablante y oyente durante este proceso.

Mientras la sintaxis estudia sólo las reglas y principios sobre cómo construir expresiones interpretables semánticamente, la semántica trata sobre el significado atribuible a expresiones sintácticamente bien formadas.

3. Estado de la técnica

En la actualidad existen pocos sistemas y prototipos comerciales que realicen la transcripción del habla [Ferrández, et al. 2004]. Dichos sistemas se suelen dividir en dos tipos:

- **Reconocedores:** Sistemas que convierten la locución de un discurso en texto escrito.
- **Sintetizadores:** Sistemas que dado un texto escrito lo transforman en una locución en la lengua destino.

El presente proyecto se limitará al estudio de los sistemas de reconocimiento del habla continua.

La mayoría de estos trabajos se desarrollan fundamentalmente en el marco estadístico [Morales, et al. 2007], pues ha demostrado ser el más adecuado para formular el problema de la traducción del habla. Típicamente todos los sistemas utilizan para ello los Modelos Ocultos de Markov.

El problema del reconocimiento del habla se puede resumir del siguiente modo:

Suponiendo que *x* es una representación acústica de un discurso dada en un idioma determinado, la traducción de *x*, en este marco consiste en encontrar la secuencia de palabras que maximizan la probabilidad de que el resultado *s* sea una transcripción literal de *x*.

En las siguientes secciones se describen trabajos relacionados con el objetivo que se persigue con este proyecto. Dichos sistemas constituyen desarrollos probados, y que han sido acogidos y aceptados por la industria. Lamentablemente existen pocas publicaciones que permitan comparar el rendimiento de cada uno de ellos [Chunrong, et al.].

Desarrollos recogidos y descritos en este proyecto son los siguientes:

- CMU Sphinx Speech Recognition System.
- HTK Speech Recognition Toolkit.
- Dragon Naturally Speaking.

Previamente en el apartado "Reconocimiento automático del habla" se introduce el funcionamiento de un ASR básico.

3.1 Reconocimiento automático del habla

El reconocimiento automático del habla (ASR) se refiere a un proceso por el cual un sistema informático (u otro tipo de máquina) identifica palabras habladas.

Existen distintos tipos de reconocedores del habla, los cuales se detallan en la sección "Tipos de reconocimiento del habla". Posteriormente se describe el esquema básico de funcionamiento de un ASR y se detallan los modelos fonéticos y del lenguaje que utilizan dichas tecnologías.

3.1.1 Tipos de reconocimiento del habla

Las tecnologías de reconocimiento del habla pueden ser separadas en diferentes clases en función del tipo de *utterances* que son capaces de reconocer.

Una *utterance* se puede definir como la vocalización de una o varias palabras que representan un significado único para el sistema. Pueden estar formados por una palabra, varias palabras, una sentencia o varias sentencias.

Esta división en clases se fundamenta en que una de las mayores dificultades a la hora de diseñar un ASR es la habilidad para determinar cuando el hablante comienza o finaliza una *utterance*.

Los ASR se pueden dividir en los siguientes tipos [Cook, 2002]:

Palabras aisladas

Los reconocedores de palabras aisladas normalmente requieren que cada *utterance* posea silencios en ambos lugares de la ventana de la muestra. Esto no significa que reconozca únicamente palabras individuales, sino que requiere una *utterance* cada vez. Normalmente estos sistemas tienen estados de "escucha/no-escucha", que requieren que el hablante espere entre *utterances*.

Palabras conectadas

Los sistemas de palabras conectadas son muy similares a los sistemas de palabras aisladas, pero a diferencia de estos, los de palabras conectadas permiten separar *utterances* con una pausa mínima entre ellas.

Habla continua

Los reconocedores de habla continua están más evolucionados y requieren de la utilización de métodos especiales para determinar los límites de las *utterances*. Permiten a los usuarios hablar de manera natural, mientras el sistema determina el contenido.

Habla espontánea

Existen diferentes definiciones acerca de que se puede considerar habla espontánea. Se puede definir como un discurso natural y no ensayado. Un sistema ASR de habla espontánea debe ser capaz de manejar una variedad de características del habla naturales como palabras que se ejecutan en conjunto e incluso ligeros tartamudeos.

3.1.2 Esquema básico de un ASR

Un sistema de reconocimiento del habla básico está compuesto por los elementos mostrados en la Figura 4.

Como se puede observar, en el diagrama de la figura se muestran dos tipos de modelos: los modelos acústicos y los modelos del lenguaje.

- Los modelos acústicos incluyen la representación del conocimiento acerca de acústica, fonética y la variabilidad del entorno. También representan el conocimiento acerca de las diferencias entre el género y el dialecto del hablante.

- Los modelos del lenguaje (que incluyen a los modelos fonéticos y lingüísticos que serán explicados más adelante) se refieren al conocimiento del sistema sobre que constituye una posible palabra, que palabras pueden ir juntas y en que secuencia.

En el reconocimiento del habla existen incertidumbres relacionadas con:

El estilo particular del habla, el reconocimiento de los segmentos básicos del habla, la variedad de palabras posibles, la existencia de palabras desconocidas, las distintas variaciones gramaticales, la presencia de ruido u otras interferencias y el acento de personas no nativas.

Un sistema completo de reconocimiento del habla debe gestionar dichas incertidumbres.

La siguiente figura presenta el esquema general de funcionamiento de un ASR básico:

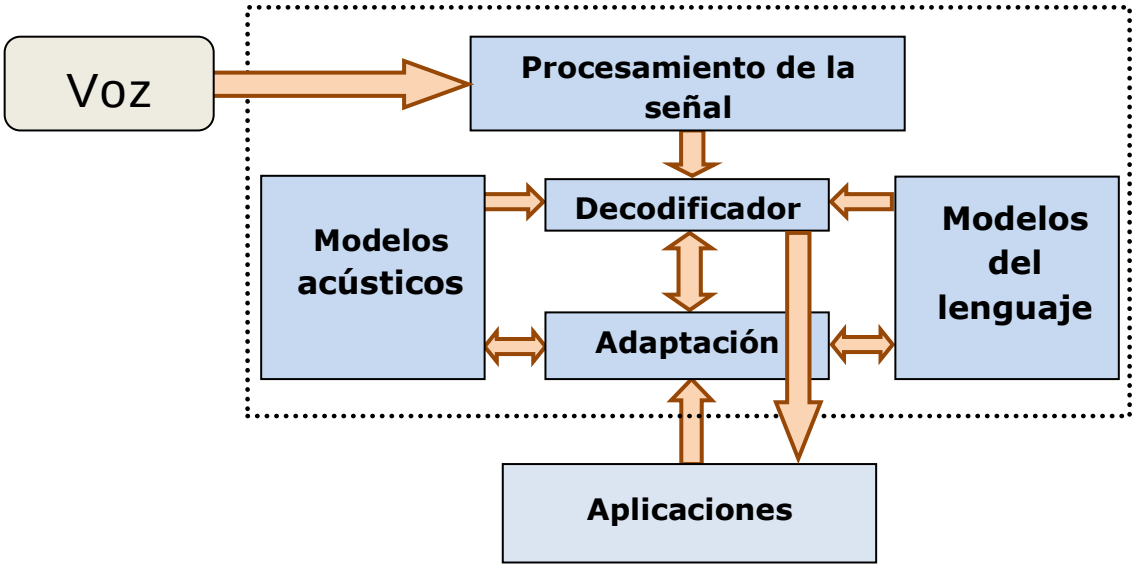


Figura 4 - Esquema básico de un ASR

Como se puede observar, la señal del habla es procesada en el denominado módulo de procesamiento de la señal que extrae una colección de vectores de características que serán utilizados por el decodificador.

Dicho decodificador usa modelos acústicos y del lenguaje para generar una secuencia de palabras las cuales posean la máxima probabilidad para los vectores de características proporcionados en el paso anterior.

En el proceso de adaptación se modifican dichos modelos a partir de la evaluación de la exactitud de la transcripción.

3.1.3 Modelos fonéticos

Un modelo fonético es un modelo estadístico que permite identificar con una probabilidad determinada la ocurrencia de un fonema en una locución o discurso.

Existen dos fases diferenciadas en la generación de un modelo fonético: la primera está relacionada con la extracción de características de la señal de voz y la segunda implica el uso de dichas características para la identificación de fonemas.

Extracción de características

El proceso de extracción de características se realiza generalmente en el dominio de la frecuencia.

Este proceso resulta fundamental para el correcto funcionamiento de un ASR por dos motivos:

- Es necesario extraer la información dentro de la señal que es más importante para la discriminación de patrones entre diferentes clases. Una buena extracción es aquella que resalta las similitudes dentro de cada clase y aumenta las diferencias entre clases diferentes.

- Es importante disminuir la cantidad de datos para que la manipulación de patrones sea computacionalmente factible. La señal de voz suele tener un régimen binario (del orden de Kilobytes/segundo) que es demasiado grande para ser manejado por un sistema de reconocimiento del habla básico.

Las características que se extraen suelen estar relacionadas con el espectro instantáneo de la señal de voz o la forma instantánea del tracto bucal.

Dado que las articulaciones que se utilizan en la producción de la voz cambian muy lentamente, es suficiente la extracción de características en intervalos de 10 a 20 ms.

Entrenamiento y reconocimiento

A partir de las características extraídas en la fase anterior, se construyen una serie de modelos estadísticos con los cuales se identificarán posteriormente con cierta probabilidad fonemas en otras locuciones. Existen diferentes técnicas para realizar este proceso:

- **HMM (Hidden Markov Model):** Basado en la creación de modelos de fonemas en estados. Esta técnica se describe con mayor detalle en el apartado "Modelos ocultos de Markov (HMM)".

- **DTW (Dynamic Time Warping):** Consiste en alinear de manera temporal los parámetros del archivo de test y los parámetros de los modelos, obteniendo de este modo la función que alinea a ambos cuando se elige la función de menor coste posible para dicha adaptación.

3.1.4 Modelos del lenguaje

Un modelo del lenguaje (en adelante LM, siglas del término inglés Language Model) en reconocimiento del habla sirve para tratar de predecir la siguiente palabra en una secuencia hablada a partir de las palabras anteriormente identificadas.

La principal ventaja de su uso es que dado un historial de palabras anteriores identificadas en una frase, el número de palabras que no se deben considerar como palabras próximas es mucho menor que el tamaño del vocabulario.

Se puede definir entonces un LM como una descripción estocástica de probabilidades de texto de *n* palabras consecutivas en los textos de entrenamiento. Los valores típicos de *n* son 1, 2 (bigramas), y 3 (trigramas).

Al tratarse de una descripción estocástica es muy común la integración de LM con Modelos ocultos de Markov (se detallan en el siguiente apartado).

3.1.5 Modelos ocultos de Markov (HMM)

Los modelos ocultos de Markov (en adelante HMM, siglas del término inglés Hidden Markov Model) son un método estadístico de caracterización de las muestras de datos observados en series de tiempo discretas. Las muestras de datos obligatoriamente tienen que poder ser distribuidas en las series de tiempo de manera continua o discreta. Dichas muestras de datos pueden ser representadas como escalares o vectores.

La teoría básica del método de HMM fue publicada en una serie de artículos clásicos publicados por Baum et al. Desde entonces se ha convertido en uno de los métodos estadísticos más poderosos en cuanto al modelado de señales del habla [Huang, et al. 2001].

Los HMM también pueden entenderse como una máquina de estados finita en la que las observaciones son una función probabilística del estado, siendo un proceso doblemente estocástico formado por:

- Un proceso estocástico oculto no observable directamente que se corresponde con las transiciones entre estados.

- Un proceso estocástico observable cuya salida es la secuencia de valores espectrales.

Una razón por la que los HMM se utilizan en el reconocimiento de fonemas es que una señal de voz puede visualizarse como una señal invariante a corto plazo (de una duración de 10 ms a 20 ms). La voz se podría interpretar entonces como un HMM para muchos procesos estocásticos (que son conocidos como estados).

Una ventaja de usar este método es que evita la limitación que posee DTW de no poder realizar un entrenamiento estadístico ya que esta técnica realiza comparaciones entre secuencias de vectores de parámetros.

Elementos de HMM

Se supone un HMM discreto en el que las observaciones posibles pertenecen también a un conjunto discreto. De manera formal los elementos de dicho HMM pueden definirse de la siguiente manera:

- **N**: número de estados del modelo, en el que q_t denota el estado en el instante de tiempo *t*. En la siguiente ecuación *S* denota el conjunto de estados del modelo.

  $$S = \{s1, s2, ..., sN\}$$

- **M**: Dimensión del conjunto de observaciones distintas de salida (el tamaño del alfabeto). En la siguiente ecuación *V* denota el conjunto de observaciones distintas de salida.

  $$V = \{v1, v2, ..., vM\}$$

- **A=$\{a_{ij}\}$**: Matriz de probabilidad de transición entre estados, donde a_{ij} es la probabilidad de transición del estado *i* al estado *j*.

  $$a_{ij} = P(s_t = j \mid s_{t-1} = i)$$

- **B=$\{b_i(k)\}$**: Matriz de probabilidad de emisión de símbolos donde $b_i(k)$ es la probabilidad de emitir un estado o_k cuando se alcanza el estado *i*. Si se considera **X**=X_1, X_2,...,X_t como la salida observada, la ecuación puede reescribirse de la siguiente manera:

  $$b_i(k) = P(X_t = o_k \mid s_t = i)\Pi$$

- **Π = $\{\pi_i\}$:** Distribución del estado inicial donde:

  $$\pi_i = P(s_0 = i) \qquad 1 \le i \le n$$

De esta forma se define un HMM como la siguiente tupla:

$$\lambda = (A, B, \pi)$$

Dada esta definición surgen tres problemas que es necesario resolver para que los HMMs tenga utilidad en aplicaciones reales:

1. **Problema de evaluación de la probabilidad:** Dado un modelo λ y una secuencia de observaciones **X**=X_1, X_2,...,X_t, encontrar la probabilidad $P = (X \mid \lambda)$ de que el modelo genere las observaciones. Esto se consigue con el algoritmo forward-backward.

2. **Problema de encontrar la secuencia de estados óptima**: Dado un modelo λ y una secuencia de observaciones, encontrar una secuencia de estados **S**=S_1, S_2,...,S_t en la que el modelo produzca las observaciones. Esto se consigue a través del algoritmo de Viterbi.

3. **Problema de entrenamiento del modelo (Problema de aprendizaje):** Dado un modelo λ y una secuencia de observaciones, se debe ajustar el parámetro λ para maximizar la probabilidad $P = (X \mid \lambda)$. Esto se consigue a través del algoritmo de Baum-Welch.

3.2 CMU Sphinx Speech Recognition System

El sistema de reconocimiento del habla Sphinx fue desarrollado por la Carnegie Mellon University (CMU), Hewlett Packard (HP) y los laboratorios de investigación de Sun Microsystems y Mitsubishi Electric (MERL) [Hopfe, et al. 2010]. La última versión de la plataforma Sphinx es Sphinx-4.

A continuación se expone una breve reseña de las cuatro versiones de Sphinx que existen actualmente [Singh, 2003]:

- **Sphinx-1:** Escrito en el lenguaje de programación C y considerado como uno de los primeros sistemas de Reconocimiento Automático del Habla de alto rendimiento independientes del locutor. Conseguía un acierto de aproximadamente un 90% en tareas de reconocimiento con vocabularios de 1000 palabras. Se basaba en el uso de modelos ocultos de Markov empleando distribuciones discretas.

- **Sphinx-2:** Basado en modelos ocultos de Markov semicontinuos y escrito en lenguaje de programación C como su antecesor.

- **Sphinx-3:** Este fue el último de los sistemas Sphinx escrito en el lenguaje de programación C. Empleaba modelos ocultos de Markov de densidad continua en un espacio de vectores continuo.

- **Sphinx-4:** Incorporaba los últimos avances en reconocimiento del habla multimodal y posibilitaba su despliegue en una amplia gama de dispositivos (incluso móviles).

La versión actual denominada Sphinx-4 incorpora mejoras respecto a los anteriores sistemas *Sphinx* en términos de modularidad, flexibilidad y otros aspectos de algorítmica que incluyen innovaciones las cuales permiten incorporar múltiples fuentes de información de forma elegante [Lamere et al. 2004].

El sistema fue enteramente desarrollado en el lenguaje de programación Java, y es altamente portable y flexible, a la vez que es sencillo el hacer que funcione con varios hilos de ejecución al mismo tiempo. Al estar desarrollado en Java posee ventajas inherentes en cuanto a mantenimiento de código se refiere.

Las versiones de Sphinx escritas en el lenguaje de programación C en cambio ofrecen un rendimiento muy superior a las escritas en Java [Anduaga, et al. 2006].

Sphinx-4 es altamente modular y flexible, soporta todos los tipos de modelos acústicos basados en HMM y todos los tipos de lenguajes de modelado y usa múltiples estrategias de búsqueda [Lamere et al. 2004]. Las innovaciones algorítmicas que incluye permiten el incorporar múltiples fuentes de información de una manera más elegante comparado con otros sistemas de la familia Sphinx.

Sphinx-4 es un sistema licenciado cómo código libre, el cual es público y está disponible en SourceForge (Carnegie Mellon University 2008).

3.2.1 Arquitectura

La arquitectura de Sphinx-4 ha sido diseñada para facilitar la modularidad. Cualquier módulo del sistema puede ser sustituido sin tener que efectuar modificaciones en cualquiera de los otros módulos.

La introducción de los módulos en la plataforma se efectúa en tiempo de ejecución, y no es necesaria la recompilación del código.

El sistema se basa principalmente en el módulo *front-end* y en el bloque de decodificación. El bloque de decodificación a su vez se descompone en tres

componentes: el gestor de búsqueda, el de lingüística y el tanteador acústico. Dichos módulos trabajan conjuntamente para realizar la decodificación.

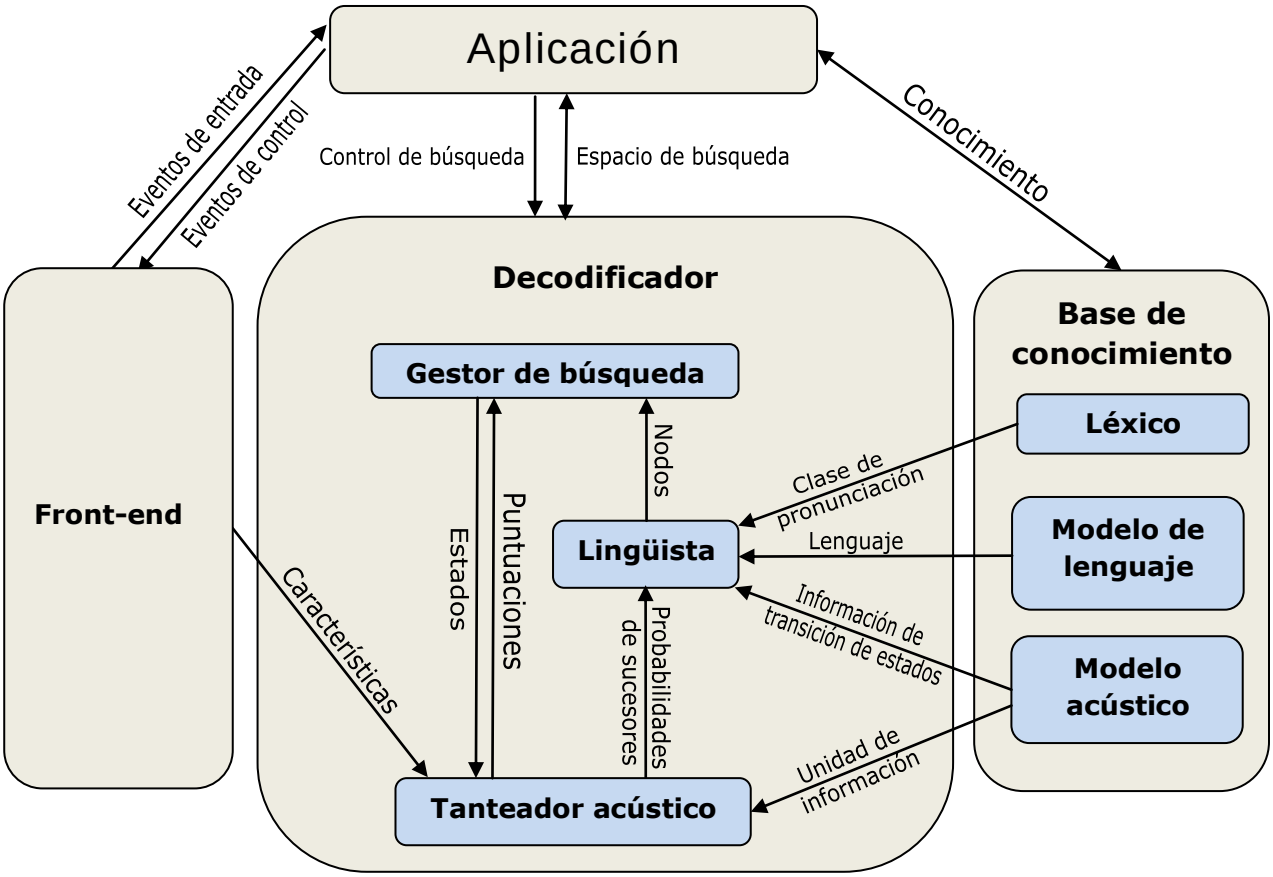


Figura 5 - Arquitectura de CMU Sphinx-4

Módulo front-end

El módulo *front-end* consiste en varios bloques que se comunican entre sí. Cada bloque tiene su entrada de datos asociada a la salida de su predecesor. Cuando un bloque está listo para recibir más datos, los lee de su predecesor, e interpreta si los datos obtenidos se refieren al habla o si son señales de control. Las señales de control pueden indicar el comienzo o el final del habla (este dato es de suma importancia para el decodificador), o pueden indicar datos borrados u otros problemas.

Si los datos se refieren al habla, son procesados y la salida se almacena en un buffer a la espera de que el siguiente bloque los solicite.

El sistema permite tres modos de funcionamiento:

- **Automático:** El sistema determina cuando comienza y termina un segmento de voz.
- ***Click-to-talk*:** El usuario indica cuando comienza a hablar, pero el sistema determina automáticamente cuando finaliza.
- ***Push-to-talk:*** El usuario indica cuando comienza y cuando termina de hablar.

La detección automática de la finalización del habla se realiza comparando el nivel de energía de la señal y tres niveles de umbral. Dos son usados para determinar el comienzo del discurso y uno para determinar el final. De este modo el sistema es capaz de detectar el comienzo y el final del habla y enviar al decodificador sólo los segmentos que contienen señal de voz.

De este modo el decodificador puede evitar desperdiciar tiempo procesando segmentos que no contienen partes del discurso del locutor.

Decodificador

El bloque decodificador se compone de tres módulos: El gestor de búsqueda, el de lingüística y el tanteador acústico. A continuación se describen detalladamente cada uno de ellos.

Gestor de búsqueda

La función principal del gestor de búsqueda es construir un árbol de posibilidades para la mejor hipótesis.

El gestor de búsqueda se comunica con el tanteador acústico para obtener puntuaciones acústicas para los datos entrantes. También se comunica con el módulo lingüista para obtener información que posteriormente utilizará para construir el árbol de búsqueda.

El gestor de búsqueda utiliza un árbol de muestra consistente en un conjunto de muestras que contienen información acerca de la búsqueda y proporcionan una historia completa de todas las rutas activas en la búsqueda.

Cada muestra contiene puntuaciones acústicas y del lenguaje para un punto dado, una *frase HMM* de referencia, una trama de entrada identificativa y una referencia a la muestra anterior para permitir una vuelta atrás.

La frase HMM de referencia permite al gestor de búsqueda el categorizar totalmente cada muestra.

El algoritmo de búsqueda mantiene una lista de muestras activas en una estructura de datos denominada *lista activa*. Dicha lista representa los extremos de las ramas activas de búsqueda. Durante la búsqueda, cada trama de características es tanteada usando modelos acústicos asociados a cada muestra en la lista activa y las ramas menores en el tanteo son podadas. El gestor de búsqueda actualiza la lista activa basándose en el estado de las frases HMM de las muestras del sucesor antes de la poda.

Sphinx-4 utiliza un algoritmo de poda que limita los resultados a un mínimo relativo configurable para la mejor puntuación al tiempo que mantiene el número total de símbolos activos a un máximo configurable. También usa un recolector de basura que elimina las muestras no utilizadas.

Cada unidad es descompuesta en estados HMM. La frase HMM comprende todos estos estados, que son conectados por arcos que poseen probabilidades del lenguaje, acústicas y de inserción.

Tanteador acústico

El cometido del tanteador acústico es calcular estados de probabilidad de salida o valores de densidad para distintos estados para un vector de entrada dado. El tanteador acústico proporciona estos resultados bajo demanda al gestor de búsqueda.

Para calcular estos resultados, el tanteador debe comunicarse con el módulo *front-end* y obtener una colección de características.

El tanteador mantiene toda la información perteneciente a las salidas de densidades de estado. El gestor de módulo desconoce si el tanteo se ha realizado usando HMM continuos, HMM semicontinuos o HMM discretos.

Módulo de lingüística

El módulo de lingüística traduce las restricciones lingüísticas proporcionadas al sistema en una estructura de datos interna denominada gramática la cual es utilizada por el gestor de búsqueda.

La gramática es un grafo dirigido en el que cada nodo representa una colección de palabras que pueden ser expresadas en un tiempo concreto. Los nodos se conectan por arcos los cuales tienen asociadas probabilidades del lenguaje que se utilizan para predecir la probabilidad de tránsito de un nodo a otro.

Las gramáticas pueden ser proporcionadas en tres formatos distintos; En forma de lista de palabras, en forma de una N-gramas (formato estándar de ARPA), y en forma de transductor de estados finitos. No obstante la arquitectura de la plataforma permite cargar nuevas gramáticas al sistema de manera sencilla y sin el conocimiento de las representaciones internas del espacio de búsqueda.

La gramática es compilada en una frase HMM representada en un grafo de estados dirigido en el cual cada estado representa una unidad del discurso. Esta separación entre el grafo de la gramática y el grafo de la frase HMM permite que la definición y la carga de las restricciones lingüísticas estén desacopladas de la representación interna de la frase HMM y de las operaciones del decodificador.

Los nodos de la gramática se descomponen en una serie de estados de palabras, una para cada palabra representada por el nodo.

Los estados de las palabras a su vez se descomponen en estados de pronunciación, basados en pronunciaciones extraídas de un diccionario el cual es mantenido por el lingüista. Se permiten distintas pronunciaciones. Cada estado de pronunciación es descompuesto en una serie de estados de unidad que pueden representar fonemas, difonos, sílabas, etc.

3.3 HTK Speech Recognition Toolkit

HTK (acrónimo del término inglés Hidden Markov Model Toolkit) es un conjunto de herramientas utilizadas para construir y manipular modelos ocultos de Markov. HTK se diseñó principalmente para construir procesadores del habla basados en HMM, en particular para reconocedores del habla. Por ello gran parte de la infraestructura de HTK está dedicada a esta tarea [Young, et al. 2009].

Los sistemas más exitosos en el área del reconocimiento automático del habla se basan en la utilización de la técnica del análisis estocástico [Carillo, 2007]. Aunque originalmente fue creado para aplicarlo al desarrollo de sistemas de ASR, ahora puede utilizarse en cualquier área de conocimiento, con la única restricción de que el problema a resolver pueda ser enfocado como un proceso de modelación estocástico markoviano [Cambridge, 2003].

La herramienta consiste en una colección de módulos escritos en el lenguaje de programación C. HTK está diseñado para correr con una interfaz tradicional de línea de comandos. Cada módulo posee argumentos requeridos y opcionales.

El uso de línea de comandos facilita la escritura de scripts para la automatización de procesos. También permite interactuar con lenguajes de programación de alto nivel para crear aplicaciones más complejas. Por último ayuda a la documentación y recolección de datos de experimentos y permite dedicar todos los recursos hardware a los algoritmos que subyacen del proceso de reconocimiento.

HTK fue originalmente desarrollado en el *Machine Intelligence Laboratory* del departamento de ingeniería de la universidad de Cambridge.

Basadas en HTK, existen APIs diseñadas para facilitar el desarrollo de aplicaciones experimentales para HTK a programadores experimentados (no suelen estar diseñadas para programadores novatos o personas con poca experiencia en la materia del reconocimiento automático del habla). Dentro de estas APIs se enmarca ATK, una aplicación escrita en C++, que consiste en una capa situada en la parte superior de las librerías de HTK estándar [Young, 2007].

3.3.1 Arquitectura

La funcionalidad de HTK está integrada en diferentes módulos. HTK utiliza distintas librerías que contienen un conjunto de instrucciones para lograr una función específica en la herramienta.

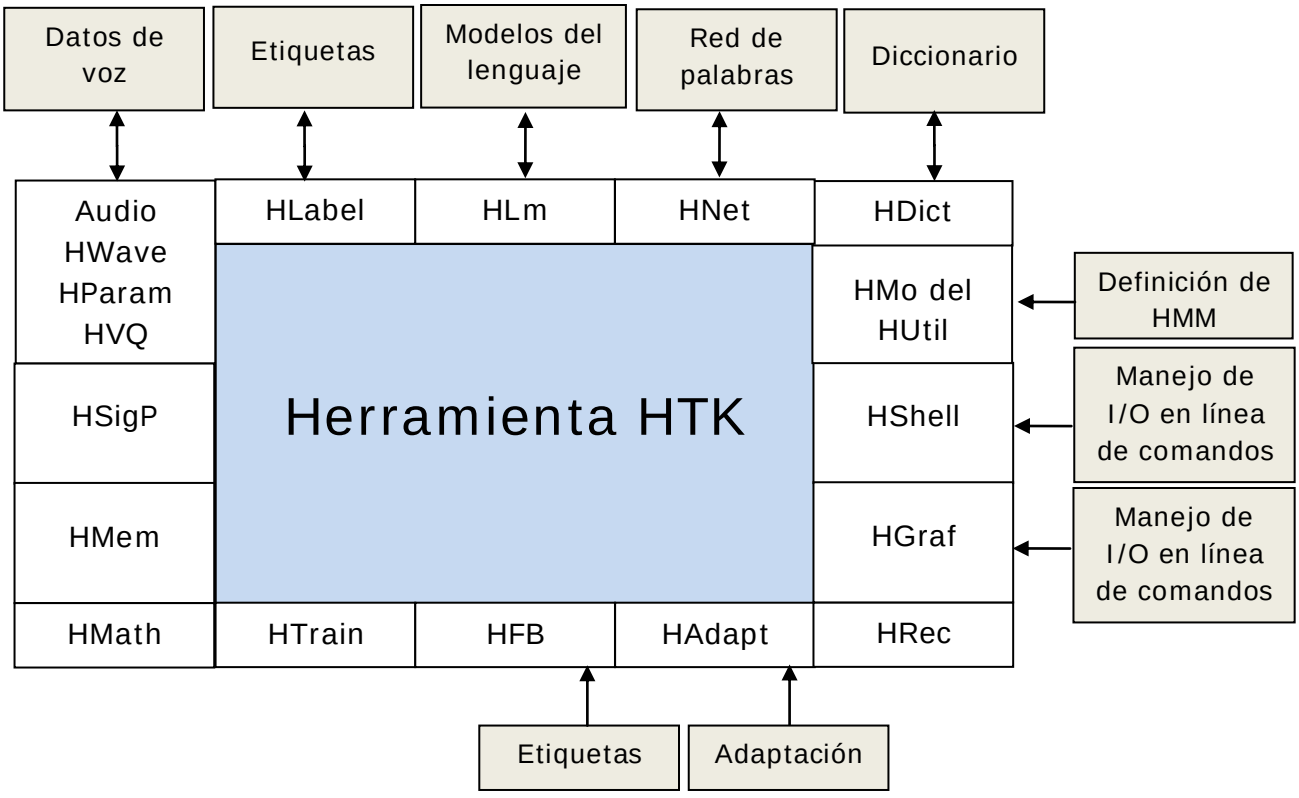


Figura 6 - Arquitectura de HTK

Los principales módulos de librerías disponibles son las siguientes:

Nombre del módulo	Funciones del módulo
HAudio	Control de la adquisición en vivo de señales de audio.
HAdapt	Adaptación del reconocedor a uno o más locutores.
HDict	Control del diccionario del reconocedor.
HGraf	Procesado gráfico de señales de audio.
HLabel	Manejo de archivos de etiquetas.
HLM	Manejo de modelos de lenguajes.
HMath	Gestión de memoria a alto nivel (Matrices y Vectores).
HMem	Gestión de memoria a bajo nivel.
HModel	Interpretación de las definiciones de HMM.
HNet	Soporte de archivos en formato Lattice y Networks.
HParm	Control de parametrización de señales de audio.
HRec	Funciones para el procesador en etapa de reconocimiento.
HSigP	Contiene operaciones para controlar alocuciones.
HShell	Interfaz entre HTK y el sistema operativo.
HTrain	Soporte para entrenamiento de modelos.
HUtil	Rutinas de manipulación de HMM.
HVQ	Manejo de libros de códigos de cuantificadores vectoriales.
HWave	Manejo de archivos de entrada y salida.

Tabla 1 - Módulos de HTK

La siguiente figura ilustra el esquema del proceso completo de reconocimiento:

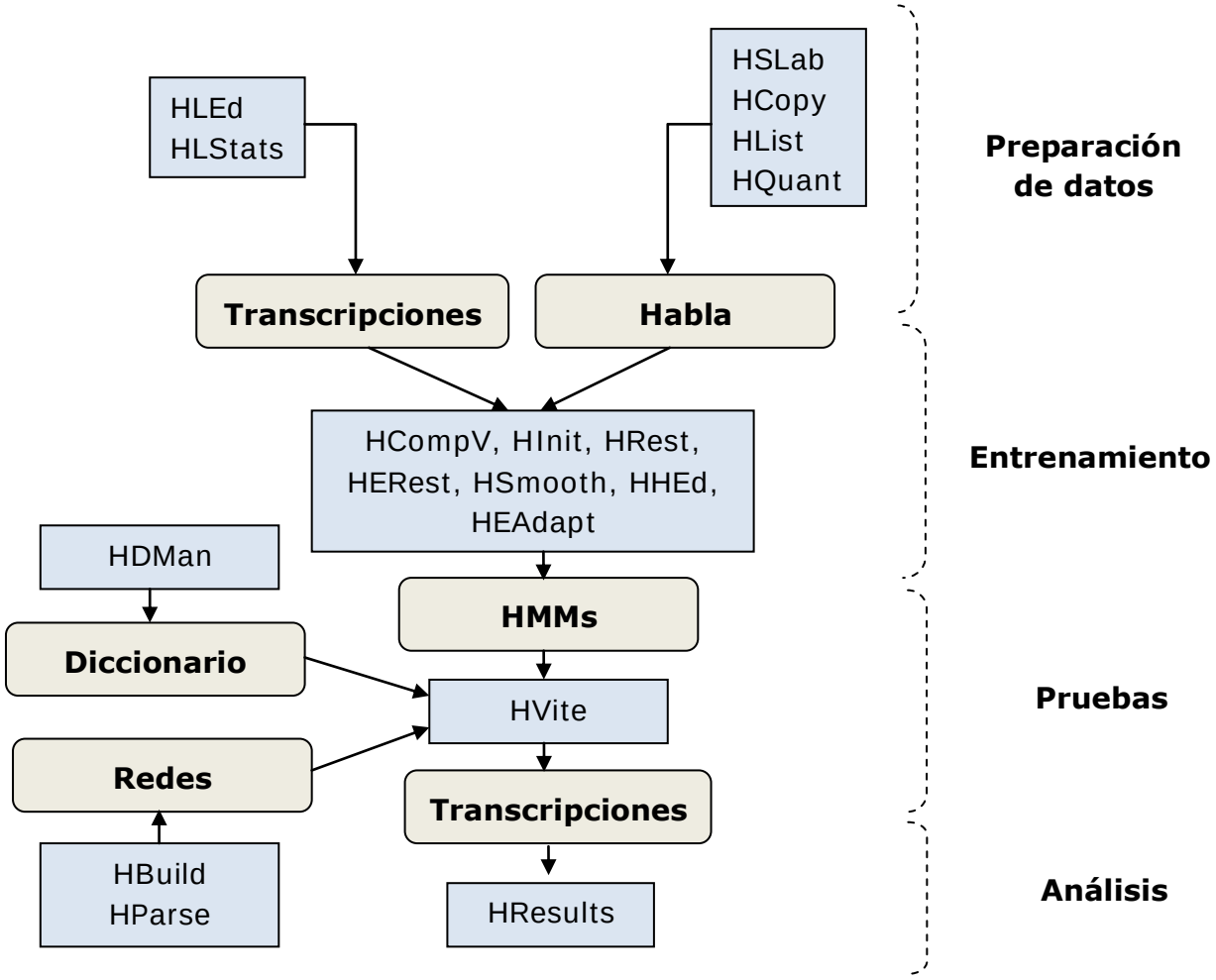


Figura 7 - Proceso de reconocimiento del habla de HTK

Como se puede apreciar el reconocimiento del habla con HTK comprende las siguientes fases:

1. Preparación de datos.
2. Entrenamiento de los modelos.
3. Prueba de reconocimiento del habla.
4. Análisis de los resultados.

En las siguientes secciones se describe detalladamente cada una de estas fases.

Herramienta de preparación de datos

La construcción de un conjunto de Modelos Ocultos de Markov necesita un conjunto de datos y sus transcripciones asociadas. Para que puedan ser usadas en el entrenamiento de modelos, deben convertirse a un formato correcto (vectores de MFFCC), del mismo modo que sus transcripciones asociadas. Si fuera necesario grabar y etiquetar con las transcripciones requeridas la voz, esto puede realizarse usando la herramienta HSLab.

Aunque todas las herramientas permiten la parametrización de ondas en tiempo real, se suelen parametrizar los datos una sola vez al principio. Para este cometido se utiliza la herramienta HCopy, que se utiliza para copiar uno o más ficheros origen en ficheros resultado.

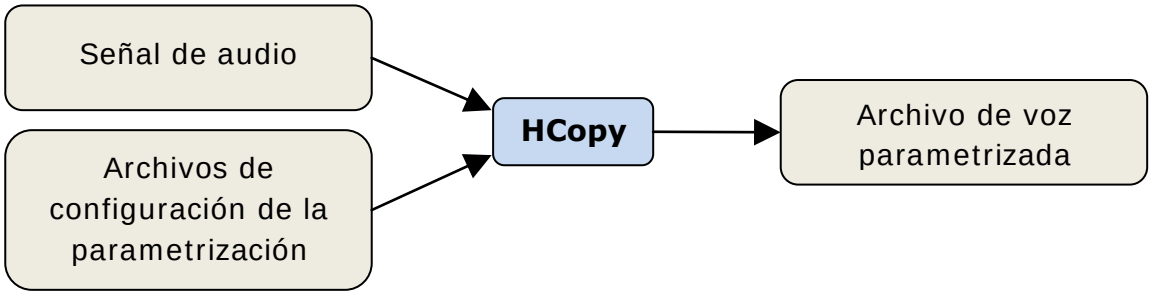


Figura 8 - Diagrama de bloques de ejecución de HCopy

HCopy incorpora mecanismos para segmentar o concatenar los ficheros origen. Antes de invocar el procesado de grandes cantidades de datos se puede usar la herramienta HList para comprobar los contenidos de cada conversión.

Figura 9 - Diagrama de bloques de ejecución de HList

Del mismo modo que las ondas de audio, las transcripciones (representadas por etiquetas) también necesitan prepararse. Para ello HLE_d implementa un editor de etiquetas que está diseñado para transformar adecuadamente los archivos que las contienen. Esta herramienta puede utilizarse para generar archivos a un único archivo de etiquetas (MLF) que es el normalmente utilizado para el procesado posterior.

Por último HStats puede recolectar y mostrar estadísticos de los archivos de etiquetas cuando así se requiera. HQuant puede ser usada para construir libros de etiquetas si fuera necesario.

Herramientas de entrenamiento de modelos

Tras la preparación inicial de los datos, es necesario definir una topología para cada HMM escribiendo la definición de prototipos. HTK facilita la construcción de HMMs con cualquier tipología. Dichas definiciones pueden ser almacenadas en ficheros de texto plano.

Se utilizan las herramientas HInit y HRest para proporcionar el entrenamiento de los modelos a partir de los datos de entrenamiento etiquetados. Cada uno de los HMMs se genera individualmente.

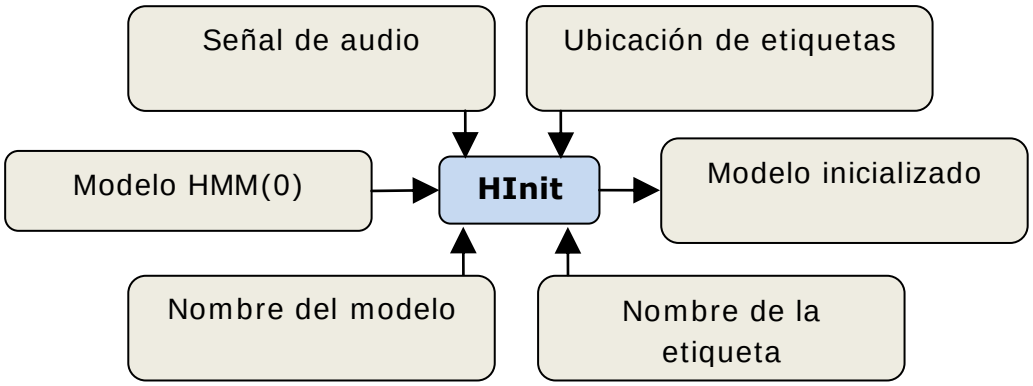


Figura 10 - Diagrama de bloques de ejecución de HInit

El diagrama en bloque de HRest es idéntico al diagrama en bloque de la llamada a HInit.

La secuencia de entrenamiento es segmentada por la herramienta HInit, y cada estado del modelo es comparado con los datos del segmento al que corresponda. De este modo se estiman las medias y las varianzas. En las siguientes iteraciones, la segmentación se afina mediante el alineamiento con el algoritmo de Viterbi [Viterbi, 1967]. La re-estimación de los parámetros se realiza mediante el método de Baum-Welch [Salcedo, 2007], ejecutado mediante la herramienta HRest.

HCOMPV permite realizar una inicialización por defecto del modelo (usando un valor global para todas las medias y varianzas).

HTK incluye una herramienta para la edición de prototipos HMM llamada HHE_D que puede ser utilizada para hacer que la transcripción dependa del hablante, de sus estados de ánimo o de otras características.

Herramientas de reconocimiento del habla

La herramienta HVITE implementa el reconocimiento del habla utilizando los modelos disponibles tras el procesado descrito anteriormente. Utiliza el algoritmo de Viterbi sobre los HMMs para encontrar el camino de estados más probable.

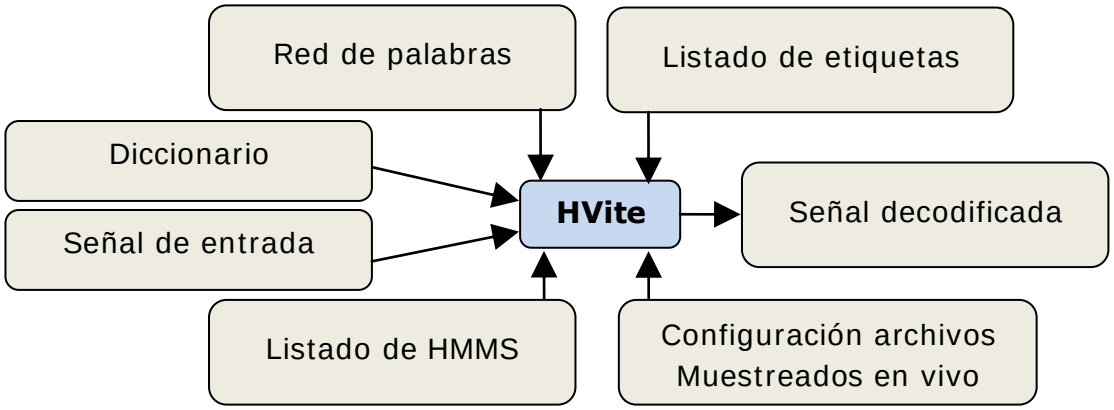


Figura 11 - Diagrama de bloques de ejecución de HVite

Dicha herramienta utiliza como entrada una red que especifica las secuencias de palabras permitidas, un diccionario que describe como se pronuncia cada palabra y un conjunto de HMMs. Funciona asignando a cada palabra desconocida un HMM según se maximice su función de verosimilitud. El reconocimiento puede hacerse tanto sobre una lista de archivos almacenado como sobre la entrada directa de audio con micrófono.

Las redes de palabras son bucles de palabras simples en las cuales cualquier palabra puede ir a continuación de cualquier otra. También puede ser un grafo con estados representando una gramática finita. Esta red describe la estructura de las frases que el reconocedor puede esperar ante una entrada desconocida.

Para crear unidades menores que la palabra e introducirlas en la red de palabras se debe utilizar la herramienta HBuild.

HLRESCORE es una herramienta que permite la manipulación de rejillas de palabras y que puede utilizarse para encontrar el camino óptimo a través de la rejilla.

Herramientas de análisis de resultados

Para evaluar el rendimiento de un reconocedor basado en HMMs es necesaria una transcripción correcta de referencia (denominada transcripción objetivo). Esta comparación se realiza con la herramienta HResults que usa métodos de programación dinámica para alinear las transcripciones resultado y objetivo y luego proporcionar estadísticos tales como sustituciones, inserciones o ausencias de palabras.

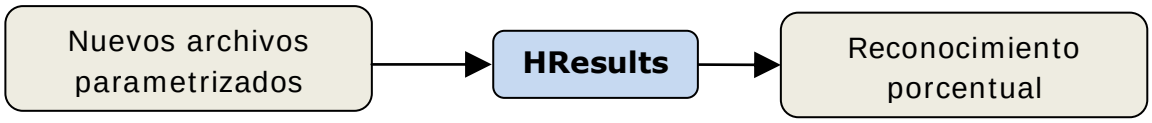


Figura 12 - Diagrama de bloques de ejecución de HResults

3.4 Dragon Naturally Speaking

Dragon Naturally Speaking es un software comercial de reconocimiento del habla desarrollado por Nuance Communications [Nuance, 2012].

En sus comienzos los sistemas Dragon sólo reconocían varios cientos de palabras presentes en discursos sencillos.

Su posterior aproximación (aproximación de Baker) utilizaba información contextual acerca de la identificación del hablante, lo que el hablante conocía, y lo que podría estar tratando de decir, aparte de las normas del habla inglesa. La aproximación de Baker estaba basada únicamente en relaciones estadísticas, como por ejemplo la probabilidad de que dos o tres palabras aparezcan tras otra en la lengua inglesa. Así crearon un diccionario fonético con los sonidos de diferentes grupos de palabras y luego comenzaron a trabajar en un algoritmo para descifrar una serie de palabras habladas basadas en las coincidencias fonéticas y la probabilidad de que una palabra se encontrara tras otra. Este enfoque pronto comenzó a superar a los sistemas de la competencia.

En 1975 ya poseían un sistema que era capaz de reconocer vocabularios de 1000 palabras sobre un IBM 370. Ya en 1990 Nuance mostró una versión para ordenador personal que reconocía vocabularios de 5000 palabras y en 1997 sacaron a la luz la primera versión de Dragon Naturally Speaking, capaz de reconocer hasta 23000 palabras. Existen versiones que funcionan sobre sistemas móviles, tales como PDA's, teléfonos móviles y tabletas. Siri, el software de reconocimiento del habla incluido de serie en el teléfono móvil iPhone 4S, fue inicialmente desarrollado por Nuance Communications y posteriormente adquirido por Apple.

James y J. Baker fundaron Dragon Systems en 1982 para lanzar al mercado productos centrados en el reconocimiento del habla.

3.4.1 Arquitectura

Al tratarse de un sistema comercial y debido a la falta de información únicamente se abordará la arquitectura implementada por las herramientas que utilizan su SDK (siglas en inglés de Software Development Kit).

El SDK de Dragon Naturally Speaking se basa en componentes ActiveX e interfaces COM basadas en la especificación Microsoft SAPI [Microsoft, 2012]. Es posible integrar a través del SDK todas las características ofrecidas por el sistema en aplicaciones usando Visual Basic, Visual C++, Delphi y otros entornos de desarrollo que soporten la especificación ActiveX.

El soporte es proporcionado por Microsoft SAPI (siglas en inglés de Speech Application Programming Interface) la cual se basa en la arquitectura en componentes COM (siglas en inglés de Component Object Model).

El SDK de Dragon Naturally Speaking se basa en la capa del SDK de Dragon que utiliza las funciones ofrecidas por su API Interna. La funcionalidad se ofrece en forma de objetos COM y ActiveX.

Nuance ofrece el SDK de Dragon en dos versiones diferentes, una para ser desplegada directamente en el cliente (Dragon SDK Client Edition) y otra para ser desplegada en un servidor (Dragon SDK Server Edition), en la cual el motor de reconocimiento en lugar de estar desplegado en la misma máquina reside en un servidor de la red.

Las funciones proporcionadas por el kit de desarrollo son bastante limitadas. No obstante permite el acceso al motor de reconocimiento del habla de Dragon, modificación del vocabulario o utilización del micrófono. A diferencia de Sphinx y HTK, no se puede acceder programáticamente al código que componen los módulos de reconocimiento del habla, ni pueden ser sustituidos.

El vocabulario del sistema Dragon consiste en un modelo del lenguaje y archivos que contienen las palabras que puede decir el usuario. El sistema posibilita la creación de vocabularios personalizados para cualquier dominio de un discurso a través de la aplicación GetCustomWords. Dicha aplicación es proporcionada por el SDK, y su cometido es obtener una lista de palabras del vocabulario del usuario y almacenar dichas palabras en un archivo.

También es posible el manejo del micrófono a través del objeto ActiveX DgnMicBtn, que permite el acceso programático al micrófono.

En la siguiente figura se describe la arquitectura utilizada por el SDK:

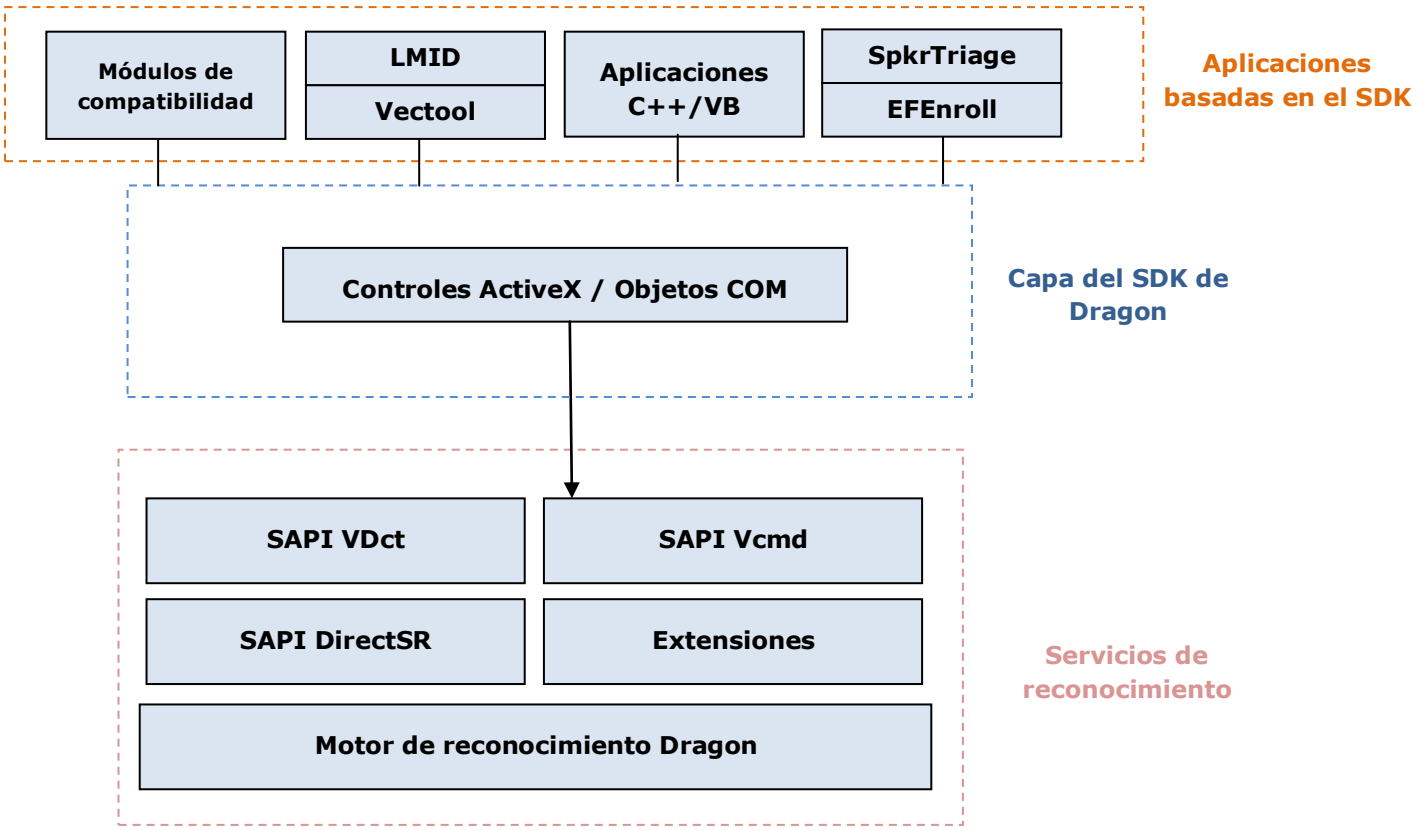


Figura 13 - Arquitectura del SDK de Dragon Naturally Speaking

Como se puede observar en la Figura 13, las aplicaciones utilizan directamente los controles ActiveX y objetos COM ofrecidos por la capa que publica las funciones disponibles en el SDK de Dragon.

Dicha capa se basa en los servicios de reconocimiento del habla, que incluyen las implementaciones basadas en las interfaces de Microsoft SAPI, el motor de reconocimiento del habla (Dragon Engine) y las distintas extensiones que emplea.

3.5 Otros reconocedores automáticos del habla

Existen infinidad de reconocedores del habla que por su falta de relevancia, por encontrarse descontinuados o por carecer de suficiente documentación no han sido objeto de estudio de este proyecto final de carrera.

A continuación se ofrece una breve descripción de algunos de ellos:

- **ViaVoice:** Fue desarrollado por IBM desde 1997 hasta 2003 y posteriomente vendido a *Scansoft,* compañía que posteriormente se fusionó con *Nuance* (propietaria de *Dragon Naturally Speaking*).

- **Linux XVoice:** Este ASR para Linux utiliza el motor de reconocimiento de *IBM ViaVoice*. La última versión fue publicada en el año 2007.

- **Verbio ASR:** Sus características se orientan al reconocimiento del habla utilizando como medio de comunicación la línea telefónica. Este ASR se utiliza actualmente en callcenters para el reconocimiento de órdenes del interlocutor. Es posible integrarlo con un sistema de verificación del locutor denominado *Verbio ASV*.

- **Microsoft Speech SDK:** Se presenta como un kit de desarrollo que permite utilizar el API Win32 Speech en aplicaciones que utilizan los lenguajes de programación Visual Basic y ECMAScript.

- **Google Voice Search:** Este producto de Google permite utilizar su motor de búsquedas empleando comandos de voz. Se puede ejecutar tanto en ordenadores de escritorio como en teléfonos móviles de última generación.

4. Proyecto Goldfinch

Este capítulo ofrece la visión general de la plataforma de experimentación y presenta las capacidades que ofrece y las restricciones impuestas a su implementación. Asimismo recoge la definición de los procesos y flujos de datos.

4.1 Introducción

En esta sección se introduce la visión general de la plataforma una vez se ha estudiado el estado de la técnica. En capítulos anteriores también se ha descrito la estructura del lenguaje hablado (el dominio sobre el que trabajará la plataforma) y el funcionamiento general de un ASR básico.

La plataforma Goldfinch debe constituir un reconocedor automático del habla continua, uno de los tipos de ASR definidos en la sección "Tipos de reconocimiento del habla".

El reconocedor se compone de un conjunto de módulos externos, que son desarrollados, seleccionados y configurados por el usuario.

A continuación se muestra un diagrama que representa la visión global del sistema:

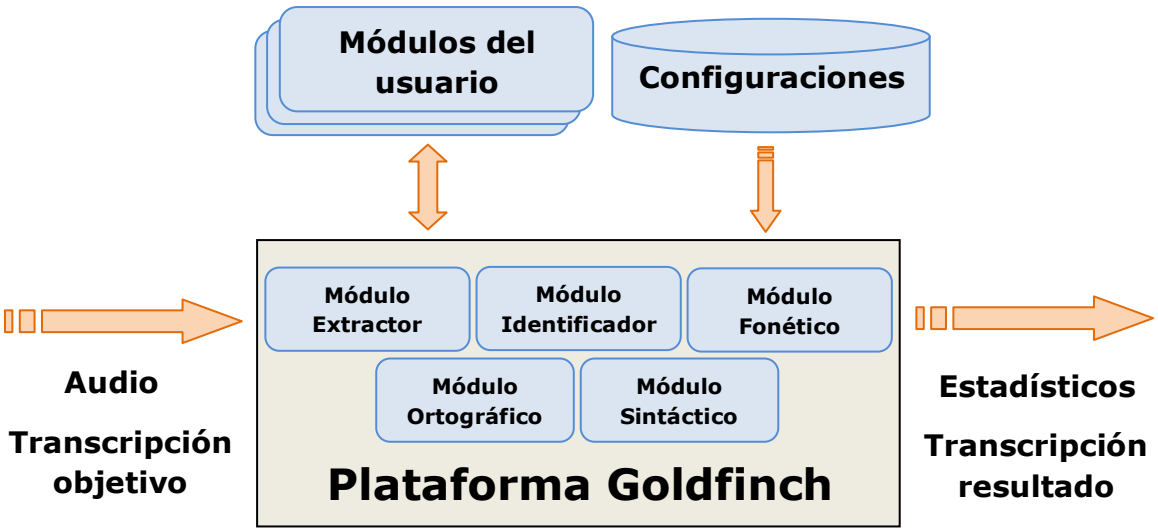


Figura 14 - Visión global del sistema

Los módulos mostrados en la Figura 14 encapsulan las funciones de los tres primeros niveles del lenguaje verbal (nivel fónico, nivel morfológico y nivel sintáctico) como procesos distintos.

Se deberá incluir también un proceso para la identificación del locutor y otro distinto para la extracción de características. El proceso de extracción de características cubrirá las funciones de procesamiento de la señal que fueron definidas en la Figura 4.

Estos cinco procesos son conceptualmente muy similares a los presentados en el tercer capítulo para los sistemas CMU Sphinx-4 y HTK.

Se pretende que los módulos que incluyen las funciones para la ejecución de los procesos anteriormente definidos sean intercambiables en tiempo de ejecución.

El sistema tiene que manejar tres tipos de entradas:

- La señal de audio.
- Transcripción objetivo asociada a la señal de audio.
- Archivos de configuración.

A partir de estas entradas devolverá en forma de texto la transcripción resultante y los estadísticos obtenidos tras comparar el resultado con la transcripción objetivo.

4.2 Capacidades y restricciones generales

En esta sección se describen las principales capacidades y restricciones del sistema. Se consideran capacidades aquellas funciones y prestaciones que proporciona el sistema. Las restricciones sin embargo recogen las limitaciones impuestas en las prestaciones del sistema.

Este apartado contiene una visión sintética y breve de los requisitos de usuario, por lo que sirve de introducción al mencionado conjunto de requisitos.

La principal prestación que ofrece el sistema es la transcripción automática del habla continua. El reconocimiento del habla será efectuado en tiempo real por los módulos que seleccione y configure el usuario. Dicha configuración puede ser introducida manualmente por el usuario, o a través de configuraciones preprogramadas almacenadas en ficheros XML.

Los módulos vendrán definidos en un conjunto de bibliotecas externas, que serán independientes de la plataforma. Dichas bibliotecas implementarán unas interfaces previamente definidas. Dichas interfaces suponen un conjunto de funciones y procedimientos comunes que deben ser implementados por los módulos que sean desarrollados por el usuario. Esto permite que el usuario pueda escoger entre los módulos que en cada momento le resulten más convenientes.

Tal y como se ha definido en el apartado anterior, los módulos configurables pueden ser de cinco tipos:

- Módulo extractor de características.
- Módulo de reconocimiento del locutor.
- Módulo fonético.
- Módulo ortográfico.
- Módulo sintáctico.

La salida de cada uno de los módulos constituye la entrada del módulo siguiente.

El sistema recibirá la señal de audio en formato PCM contenida en un fichero WAV. Una vez cargado dicho fichero, el sistema ofrecerá información al usuario acerca de parámetros de la señal de audio, tales como una gráfica de amplitud y otras características de la señal digital de audio.

La plataforma debe controlar los errores que pueda producir el usuario, mostrando mensajes de error en una consola de ejecución a modo de log. Otra información que se debe proporcionar al usuario está relacionada con las propiedades del fichero de audio y del progreso de la transcripción.

Este proyecto no contempla la definición de los modelos acústicos ni del lenguaje. Tampoco contempla la implementación final de los módulos, aunque si se deberán entregar módulos de prueba para probar el correcto funcionamiento de la plataforma.

4.3 Procesos y flujos de datos

Para la descripción de los procesos necesarios en el desarrollo de la plataforma se ha considerado oportuno utilizar el diagrama de flujo de datos (DFD); esta técnica se encuentra vinculada al denominado *Análisis Estructurado* pero posee una capacidad de descripción muy adaptada a las necesidades del proyecto.

Este tipo de diagramas representa de una manera visual el flujo de datos entre los distintos procesos, entidades externas y almacenes de datos que conforman el sistema. Los DFD también sirven para ilustrar como los procesos transforman los datos que se les proporciona como entrada en información útil [B. Shelly, et al. 2012].

El sistema ha sido divido en tres niveles para cumplir con la regla 7±2 de la descomposición modular recursiva (nivel 0 ó de contexto, nivel 1 o nivel de sistema y nivel 2 o de expansión). El nivel 0 representa el mayor nivel de abstracción, que es desarrollado por el nivel 1 y este profundizado por el nivel 2.

4.3.1 Contexto del sistema

El diagrama de contexto muestra el proceso que representa el sistema y la relación que mantiene con las entidades externas con las que interactúa.

En este proyecto la única entidad externa será el usuario de la plataforma. La única salida que proporciona el sistema es la transcripción.

El único proceso es la propia plataforma que interactúa con las entidades externas anteriormente descritas.

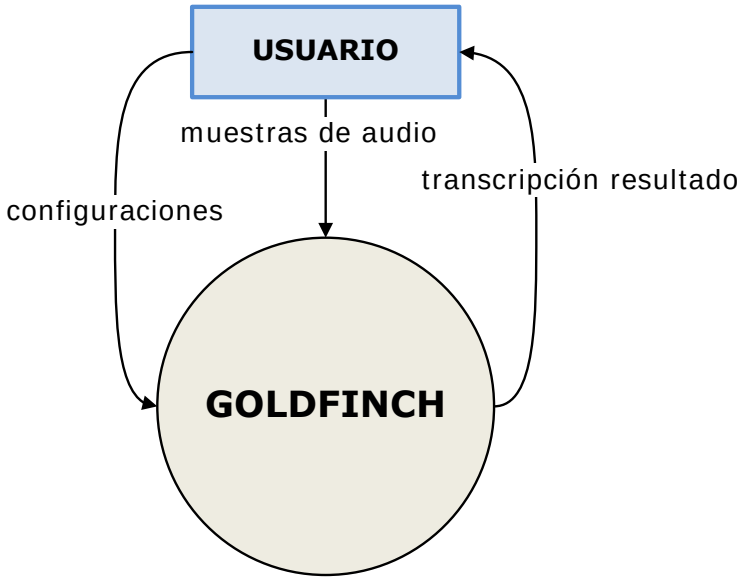


Figura 15 – Contexto del sistema

La plataforma Goldfinch se descompone a su vez en cuatro procesos más, representados en el diagrama de nivel 1 o de sistema. Estos procesos son los siguientes: Procesador de audio, reconocedor del interlocutor y ambiente, configurador y analizador.

El procesador de audio se encarga de transformar el audio en un flujo de bytes, independientemente del formato del archivo escogido. Posteriormente proporciona dicho flujo al reconocedor del interlocutor y ambiente.

El reconocedor del interlocutor y ambiente analiza el flujo de bytes y facilita al proceso analizador sus resultados, junto con el vector de bytes sin modificar.

El proceso analizador a partir del vector de bytes que recibe del procesador de audio y del modelo que utilice en ese momento devuelve unos resultados de transcripción.

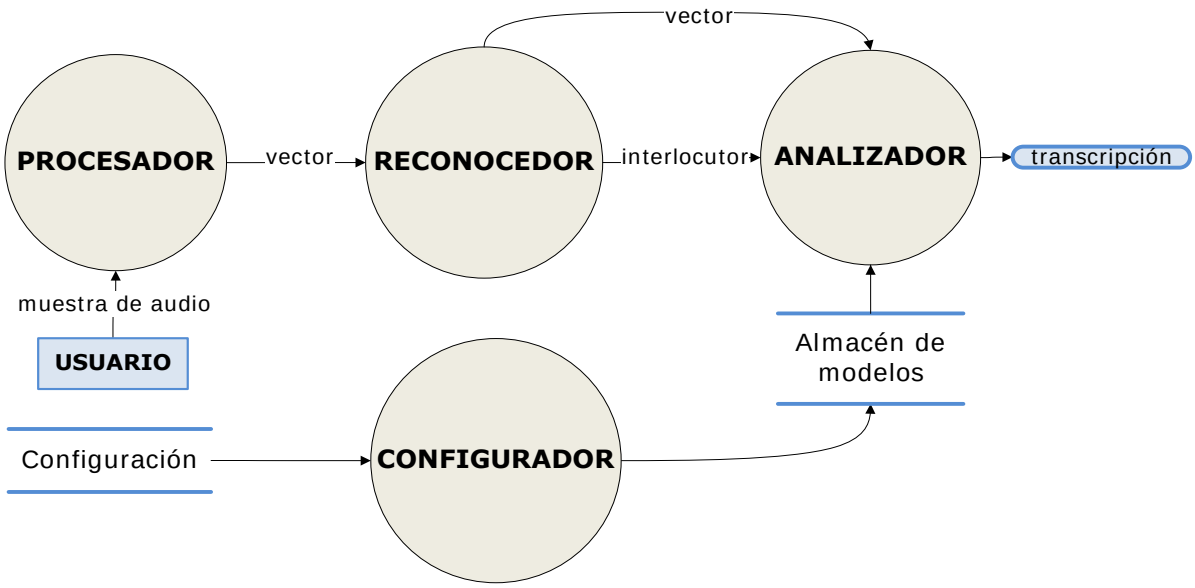


Figura 16 - Diagrama del sistema

4.3.2 Componentes Software de alto nivel

Los procesos del procesador de audio y el analizador, pueden a su vez descomponerse en más procesos. En esta sección se detallará cada uno de los subprocesos que de los que se compone cada uno. Estos subprocesos forman el diagrama de nivel 2 o de expansión.

Procesador

El procesador de audio se encarga de obtener características a partir de las muestras de audio contenidas en un fichero WAV con formato PCM. Este proceso puede a su vez descomponerse en otros dos subprocesos: Proveedor y extractor de características.

El proveedor de audio es el componente encargado de suministrar muestras de audio de una longitud predeterminada y fija a partir del audio almacenado.

El extractor de audio recibe estas tramas, las procesa y obtiene características que proporciona en forma de vector.

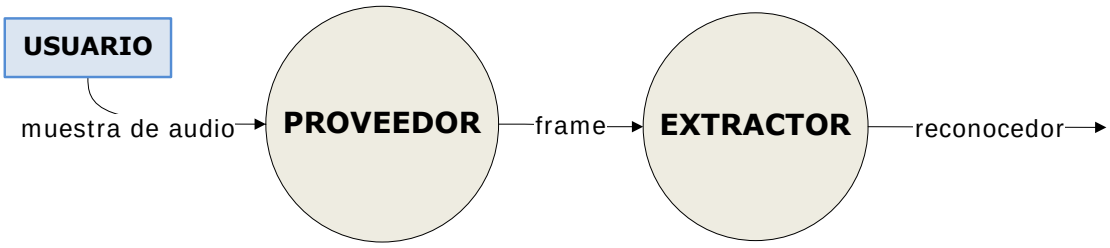


Figura 17 - Diagrama del procesador de audio

Analizador

El analizador se descompone a su vez en tres subprocesos que constituyen analizadores especializados: analizador fonético, analizador ortográfico y analizador sintáctico. Cada uno de estos analizadores se encapsula dentro de un módulo distinto.

El analizador fonético recibe un vector de características que proviene del módulo extractor (componente del procesador de audio) y utiliza un modelo fonético. A partir de estas dos entradas produce la información de los fonemas que identifica.

El analizador ortográfico recibe los fonemas del analizador fonético y utiliza el modelo ortográfico. A partir de estas dos entradas produce su salida, que está compuesta por las palabras que identifica.

Por último, el analizador sintáctico recibe las palabras identificadas por el analizador ortográfico y utiliza el modelo sintáctico. A partir de estas dos entradas genera el conjunto de frases que serán mostradas al usuario.

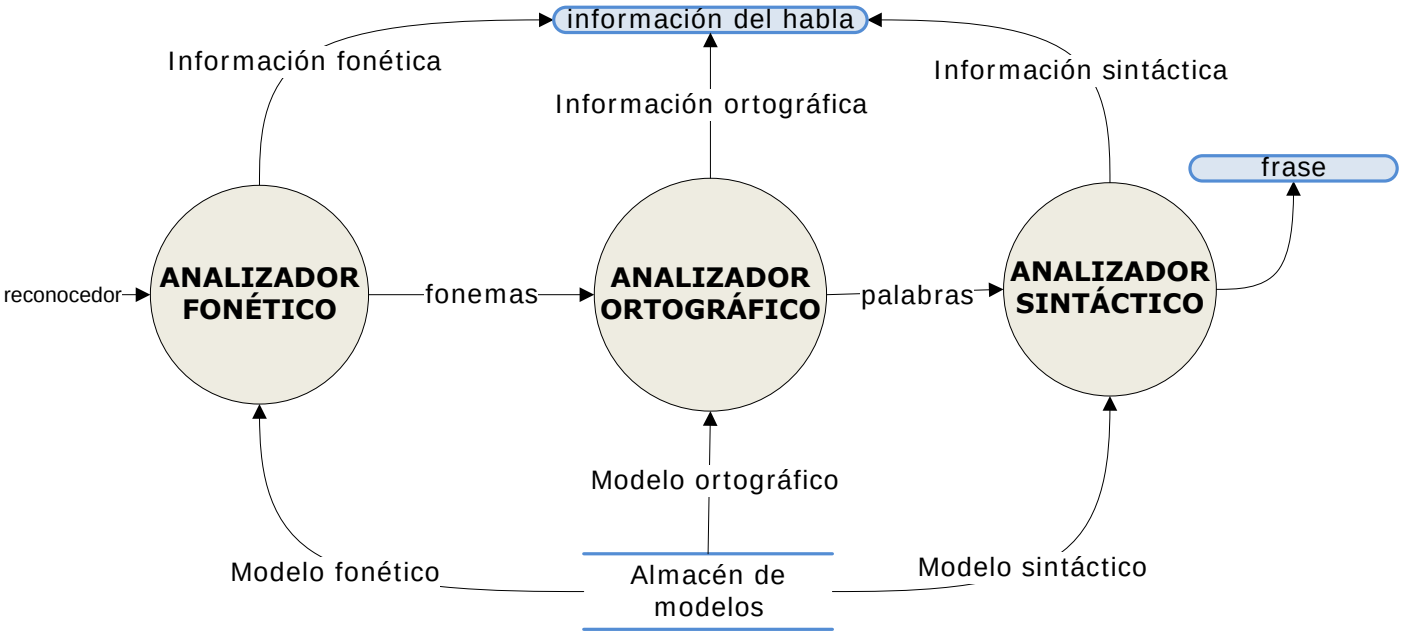


Figura 18 - Diagrama del analizador

4.4 Casos de uso

Un caso de uso describe un servicio que el usuario requiere del sistema e incluye la secuencia completa de interacciones entre el usuario y el sistema. Las actividades del sistema proporcionan servicios específicos al usuario.

En esta sección mediante los casos de uso, se describe desde el punto de vista del usuario el desempeño del sistema para ofrecer una determinada funcionalidad.

Cada una de las formas en que un usuario utiliza la plataforma está representada por un caso de uso. La importancia de los casos de uso estriba en que ofrece un valor concreto y de utilidad para el actor.

El empleo de los casos de uso facilita [Barranco, 2001]:

- Identificar quién interactúa con el sistema y qué debe hacer el sistema.
- Identificar las interfaces del sistema.
- Verificar que no se olvidan requisitos.
- Obtener un modelo de objetos del dominio inicial.

Actores

El sistema sólo contempla a un actor primario, el cual representa a la entidad externa al sistema que guarda una relación con este y al que demanda una determinada funcionalidad. El actor lo constituye un operador humano. En adelante el presente documento se referirá al actor como el *Usuario de la plataforma*.

Para la plataforma Goldfinch se contempla un caso de uso principal, que está ligado a la operación de transcripción. Dicho caso de uso incluye la ejecución de los siguientes casos de uso:

- Selección de un módulo.
- Configuración de un módulo manualmente.
- Configuración de un módulo a partir de una configuración preprogramada.
- Selección de un archivo de audio.

La configuración de módulos tanto manual como a partir de una configuración preprogramada requiere del comportamiento del caso de uso de selección de un módulo.

Existe un caso de uso adicional que contempla la obtención de los estadísticos de la transcripción. Dicho caso de precisa de la ejecución del caso de uso de realización de la transcripción.

En todos los casos las relaciones de inclusión indican que el caso de uso mencionado (caso de uso inicial) incluye el comportamiento del caso de uso final (los denominados subcasos de uso).

La Figura 19 muestra el diagrama de casos de uso de la aplicación, resaltando las relaciones de inclusión y de asociación:

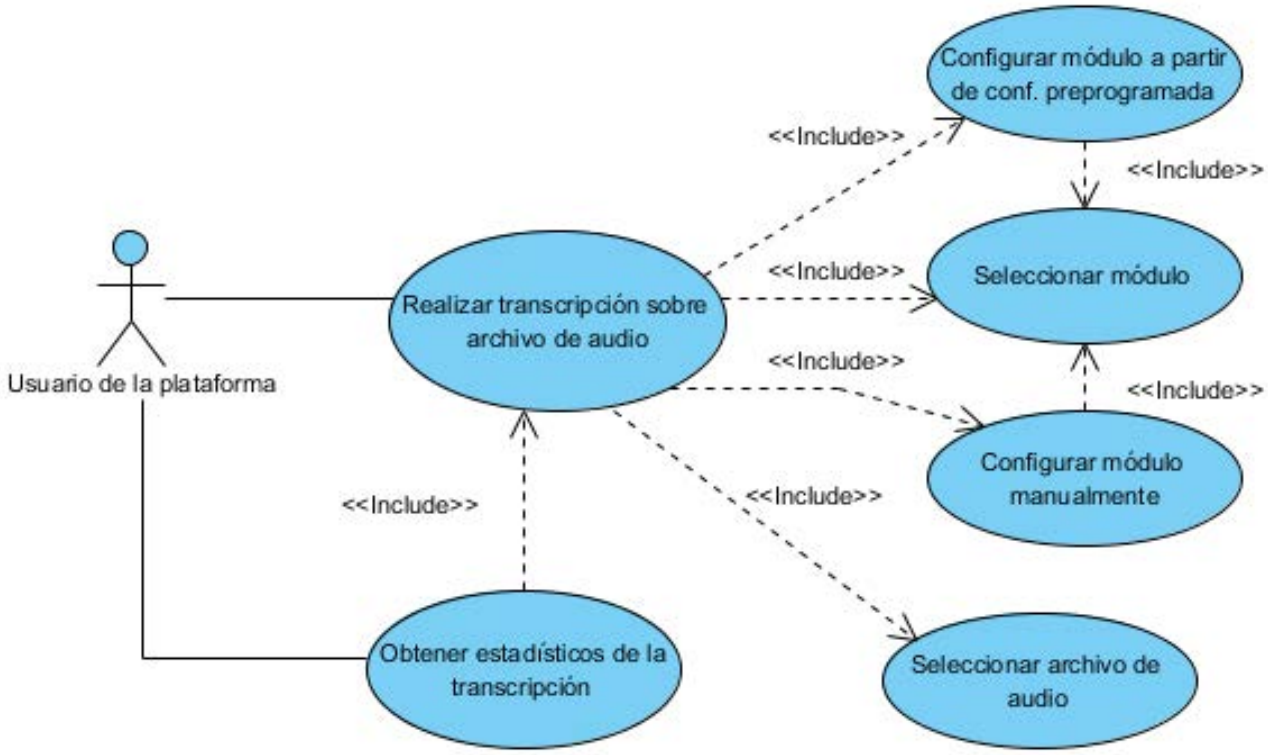


Figura 19 - Diagrama de casos de uso

A continuación se enumeran los casos de uso de los cuales se ofrece una breve reseña:

- **Seleccionar módulo:** Describe la operación de selección de un módulo por el usuario para incorporarlo a la plataforma. Esta operación debe ejecutarse al menos una vez para cada tipo de módulo.

- **Configurar módulo manualmente:** Esta operación permite al usuario seleccionar la configuración deseada a través de un formulario de introducción de datos.

- **Configurar un módulo a partir de una configuración preprogramada:** El objetivo de este caso de uso es el mismo que el del caso de uso anterior. En este caso se desea configurar un módulo a partir de los valores contenidos en un archivo en formato XML.

- **Seleccionar un archivo de audio:** Este caso de uso describe la operación de selección del archivo de audio que contiene la locución a reconocer por la plataforma.

- **Realizar transcripción del archivo de audio:** Describe el caso de uso principal de la plataforma. Como se ha explicado anteriormente, requiere de la ejecución de los casos de uso de selección del archivo de audio, selección de los módulos y configuración de los módulos. El objetivo de este caso de uso es obtener la transcripción en texto a partir de la señal de audio contenida en el archivo seleccionado.

- **Obtener estadísticos de la transcripción:** Esta operación permite evaluar los módulos seleccionados por el usuario. Requiere que el usuario seleccione previamente un archivo de texto plano que contenga la transcripción objetiva de la locución contenida en el fichero.

En el apéndice III se incluye una descripción formal de los casos de uso.

4.5 Requisitos de software

En esta sección se presentan los requisitos de software que describen como deben satisfacerse los requisitos de usuario y cuál será el comportamiento del sistema que se va a desarrollar.

En el Apéndice II se incluye el catálogo de requisitos de software de manera formal y con un nivel más amplio de detalle, incluyendo las propiedades que los definen. También se incluye la matriz de trazabilidad con los Requisitos de Usuario.

A continuación se incluye una breve descripción de los requisitos de software:

La aplicación será programada usando el lenguaje de programación Java. Por este motivo la plataforma precisará para su ejecución de la presencia de la máquina virtual de Java en el sistema. Del mismo modo la interfaz gráfica de usuario se realizará utilizando la biblioteca gráfica Swing de Java.

Según se muestra en los prototipos del Apéndice IV, debe existir un área de texto en la pantalla principal para mostrar los resultados de la transcripción. Del mismo modo debe existir un área de texto en la pantalla principal a modo de consola para mostrar los errores que ocurran en el sistema.

Con el objetivo de mostrar la duración de la parte transcrita, en la pantalla principal se mostrará una barra de progreso que inicialmente estará vacía y de representará llena cuando finalice la transcripción. Dicha barra actualizará su progreso cada segundo.

Como complemento a la barra de progreso se mostrará una etiqueta donde se indicará la duración del archivo de audio en formato "hh:mm:ss" (donde "hh" son horas, "mm" son minutos y "ss" son segundos).

Debe existir un área de texto en la pantalla principal donde imprimir las propiedades del fichero de audio. La información mostrada será la siguiente:

- Tipo de señal de audio.
- Frecuencia de muestreo.
- Número de bits que forman cada muestra.
- Si la muestra de audio es mono o estéreo.
- Formato interno de representación (Little Endian o Big Endian).

Existirá una ventana emergente que presentará los estadísticos de la transcripción una vez finalizada esta. Si no se han realizado la transcripción los valores de los estadísticos estarán a cero. Los estadísticos que se mostrarán serán los siguientes:

- Número de palabras añadidas.
- Número de palabras eliminadas.
- Número de palabras modificadas.
- Número de palabras acertadas.
- Exactitud de la palabra (valor entre 0 y 1).
- Tasa de error de la palabra (WER, siglas en inglés de Word Error Rate).
- Distancia de Levenshtein.

Los estadísticos se obtendrán comparando la transcripción objetivo (almacenada en un fichero de texto plano que podrá ser seleccionado desde la ventana de obtención estadísticos) y la transcripción obtenida.

La ejecución de la transcripción podrá controlarse mediante unos botones que se mostrarán en la ventana principal. Dichos botones son los siguientes: botón de inicio, botón de pausa y botón de detención de la transcripción.

Es preciso que se puedan incorporar al Classpath de Java en tiempo de ejecución.

Para poder iniciar la transcripción es preciso seleccionar tanto los cinco módulos de usuario como el fichero de audio que contiene la locución.

Para ello existen dos ventanas de selección de archivos:

- La ventana de selección de módulos presente en la interfaz gráfica mostrará únicamente los que contengan la extensión "*.jar*".
- Asimismo la ventana de selección de archivos de audio expondrá únicamente los que contengan la extensión "*.wav*".

En dichas ventanas de selección de archivos se presentarán las carpetas a efectos de navegación por el sistema de archivos, pero no de selección.

Una vez seleccionado el archivo de audio debe mostrarse en la pantalla principal una gráfica que represente la amplitud de la señal de audio respecto del tiempo.

Los ficheros que contienen los módulos de usuario y los archivos de audio se validarán al ser seleccionados. En ambos casos se imprmirá un mensaje de error por la consola si existe cualquier error en el formato del fichero escogido.

Si se desea cargar un módulo almacenado en un ordenador presente en la red de área local es necesario disponer de conexión de red al área local. La ruta de dicho módulo deberá estar accesible y con permisos de lectura y ejecución.

Todos los módulos desarrollados por el usuario implementarán una interfaz java que identifica la signatura de las operaciones. Se desarrollarán módulos de prueba para verificar el correcto funcionamiento de la plataforma que implementarán la interfaz citada con anterioridad.

Los módulos podrán configurarse de dos maneras:

- A través de un formulario de introducción de datos. Dicho formulario incluye validación de datos, mostrando un mensaje en la parte superior del formulario en casos de error.
- A partir de configuraciones preprogramadas contenidas en un archivo XML.

Adicionalmente los módulos podrán incluir una ruta que indique un fichero de configuración por defecto.

Es preciso que los módulos hayan sido seleccionados antes de activar los controles de configuración de los módulos.

Independientemente de la configuración de los módulos, la plataforma admitirá valores de configuración almacenados en un archivo de configuración en texto plano. Los valores de configuración estarán almacenados en pares clave-valor.

5. Decisiones Tecnológicas

En este punto se exponen las tecnologías escogidas para implementar la plataforma Goldfinch.

5.1 Frameworks, librerías y lenguajes de programación

En esta primera sección se exponen las características de los frameworks, librerías y lenguajes de programación utilizados en el desarrollo de la plataforma.

5.1.1 Java

Java es un lenguaje de programación de propósito general, que permite la concurrencia, basado en clases y orientado a objetos específicamente diseñado para tener las menores dependencias de implementación como sea posible [Gosling, et al. 2000].

Una de sus características principales es su fuerte tipado. Esta especificación distingue claramente a los errores que pueden suceder en tiempo de compilación y en tiempo de ejecución.

Las aplicaciones se traducen a un bytecode que es interpretado o compilado a código nativo para la ejecución, aunque es posible la ejecución directa del bytecode por hardware.

Las aplicaciones escritas en Java se interpretan en la máquina virtual de java (ó JVM en sus siglas en inglés) [Eckel, 2002]. La especificación de una máquina virtual de Java incluye:

- Un juego de instrucciones y la definición del significado de dichas instrucciones denominadas bytecodes.
- Un formato binario (los ficheros .class) que es usado para representar los bytecodes y la infraestructura relacionada de una forma independiente del hardware.

Un algoritmo para reconocer programas que puedan comprometer la integridad de la JVM. Este algoritmo se denomina algoritmo de verificación.

El lenguaje de programación Java se podría considerar de alto nivel, en los cuales los detalles de representación del hardware sobre el que se ejecuta no son accesibles.

Incluye un gestor de memoria denominado "recolector de basura" para prevenir los problemas de desasignación explícita de lenguajes como C ó C++.

Java fue desarrollado por Sun Microsystems a principios de los años 90. Entre diciembre de 2006 y mayo de 2007 Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU-GPL de acuerdo con las especificaciones del Java Community Process (JCP).

5.1.2 JAXB

JAXB [Oracle, 2012] es un API que permite vincular clases java con sus representaciones en XML. Su cometido es simplificar el acceso a un documento XML desde un programa Java mediante la presentación del documento XML en clases Java específicas. El primer paso de este proceso es vincular el esquema para el documento en un conjunto de clases Java (que representarán el esquema):

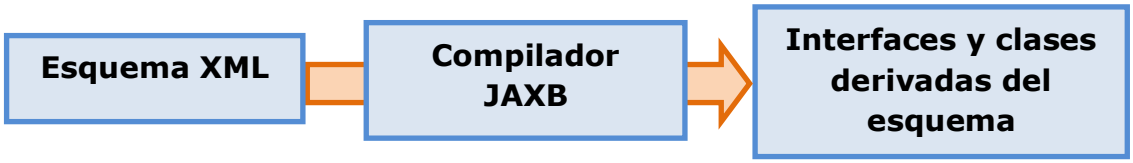


Figura 20 - Compilación de un esquema XML en interfaces y clases

Un esquema es una especificación XML que gestiona los componentes permitidos en un documento XML y las relaciones entre dichos componentes. El esquema identifica los elementos que pueden aparecer en un documento XML, especifica en qué posición deben aparecer, que atributos deben tener y que elementos son subordinados de que otros elementos. Un documento puede no tener esquema, pero si lo tiene, debe ajustarse a sus especificaciones para poder ser un documento XML válido. JAXB requiere que un documento XML tenga un esquema y que dicho esquema cumpla con el *W3C XML Schema Language* [W3C, 2003].

Conversión del documento en clases Java

La conversión del documento en clases Java genera un árbol de objetos que representa el contenido y la organización del documento. El árbol que genera JAXB es más eficiente en términos de memoria que el producido por DOM.

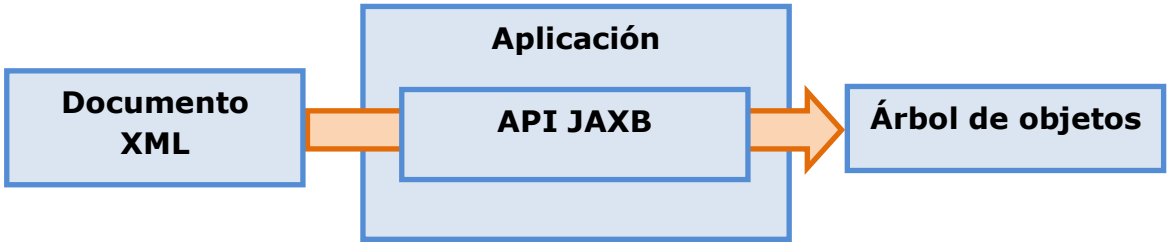


Figura 21 - Transformación de un documento XML en clases con JAXB

JAXB permite del mismo modo convertir un árbol de objetos en un documento XML (es decir, realizar la operación contraria).

Ventajas del uso de JAXB

- No es necesario crear un analizador SAX para rastrear el documento.
- Permite el acceder a los datos en orden no secuencial, y al contrario que en el procesado con DOM no es necesario el navegar a través del árbol para acceder a los datos.
- En términos de memoria es más eficiente que DOM y SAX.
- Permite la validación de los datos contra su esquema.

5.1.3 Beans Binding (JSR295)

Beans Binding es una API de código abierto que permite sincronizar las propiedades de dos objetos de Java distintos de una manera sencilla, ahorrando tiempo y minimizando la cantidad de código para escribir eventos de escucha y propagación de propiedades

[Zhelezniakov, et al.]. También soporta validación y conversión de datos, operaciones que se repiten al utilizar formularios en cualquier aplicación.

5.1.4 Joda Time

Joda Time [Colebourne, 2012] es una API de código abierto que proporciona un remplazo de las clases de fecha y hora de Java basándose en el estándar ISO8601 usado por XML. Las razones para usar Joda Time en lugar de la API de Java son las siguientes:

- Facilidad de uso.
- Soporte de distintos calendarios.
- Gran cobertura en test.
- Completa documentación.
- Madurez.

También mejora el soporte para los conceptos de intervalos, duraciones y periodos.

5.1.5 Java Sound

Java Sound [Oracle, 2012] es una API de bajo nivel para la adquisición, el manejo y la reproducción de sonido en Java. Proporciona mecanismos para instalar, acceder y manipular recursos tales como mezcladoras de audio, sintetizadores MIDI, ficheros de audio y conversores de formato de audio. No incluye editores de audio o herramientas para graficar señales, pero proporciona una base para construirlos.

Nativamente sólo acepta dos diferentes tipos de datos de audio:

- Archivos MIDI (Musical Instrument Digital Interface).
- Muestras de datos de audio (.wav).

5.1.6 Service Provider Interface

Es una característica desde J2SE 1.3 que permite a los desarrolladores añadir funciones a la máquina virtual de JAVA de forma transparente. Java Sound usa SPI en tiempo de ejecución para el implementar servicios que soporten formatos inicialmente no soportados (como por ejemplo MP3, ó Ogg Vorbis).

MP3 SPI

MP3SPI es una interfaz proveedora de servicio que proporciona soporte para el formato MP3 y que soporta streaming, ID3v2 frames, ecualización. Está basada en las librerías JLayer[3] y Tritonus.

VorbisSPI

VorbisSPI es un SPI (basado en la librería Java JOrbis) que añade soporte al formato Ogg Vorbis.

5.1.7 Apache Commons

Apache Commons [ASF, 2012] es un proyecto de la Apache Software Foundation dependiente del proyecto Jakarta[4]. El propósito de dicho proyecto es el proveer de componentes de código abierto reusables. Está compuesto por tres partes o áreas de trabajo:

- **Proper:** Dedicado a crear y mantener componentes de Java reusables.
- **Sandbox:** Donde los participantes pueden crear proyectos experimentales que no serán incluidos en el área de trabajo *Proper*.
- **Dormant:** Colección de componentes declarados inactivos por la poca actividad en su desarrollo/mantenimiento.

5.1.8 Java Swing

Java Swing [Loy, et al. 2003] es una biblioteca gráfica para Java que incluye Widgets para definir la interfaz gráfica de una aplicación de escritorio. Dichos Widgets los forman cajas de texto, tablas, ventanas, botones y otros elementos.

Se comporta como un Framework MVC en sí mismo y permite desarrollar interfaces gráficas con independencia de la plataforma en que se ejecuten.

Swing se basa en AWT que es la API estándar para suministrar una GUI en Java. Las diferencias principales con AWT es que mientras este confía en los módulos de interfaz de usuario de alto nivel del sistema operativo, Swing dibuja sus propios Widgets utilizando Java 2D para llamar a las subrutinas de bajo nivel en el subsistema de gráficos local.

[3] JavaLayer es un proyecto público y de código abierto basado en Java desarrollado por un programador conocido con el nickname "javaZOOM" y que se utiliza para decodificar archivos MP3.
[4] El proyecto Jakarta crea y mantiene software de código abierto para la plataforma Java bajo el auspicio de la Apache Software Foundation.

5.2 Herramientas Utilizadas

A continuación se exponen brevemente las herramientas utilizadas en el desarrollo del proyecto y sus características principales.

5.2.1 Eclipse IDE

Eclipse [Eclipse, 2012] es un IDE multiplataforma libre que permite la creación de aplicaciones de cualquier tipo. Fue creado inicialmente por IBM y ahora lo desarrolla y mantiene la Fundación Eclipse.

A diferencia de otros entornos monolíticos, Eclipse emplea módulos para proporcionar toda su funcionalidad, que pueden ser seleccionados por el usuario a modo de plug-ins.

Las características principales del IDE son las siguientes:

- Permite trabajar con más de un proyecto al mismo tiempo.
- Colorea el código según el lenguaje utilizado (resaltado de sintaxis).
- Proporciona información adicional sobre los errores de compilación y sugiere soluciones. Compila en tiempo real.
- Permite crear plantillas exportables para formatear el código.
- Soporta el autocompletado del código.
- Tiene diferentes herramientas que permiten la refactorización del código.
- Dispone de un repositorio central de plug-ins, y se pueden añadir repositorios adicionales.

Eclipse IDE se autodefine como "An IDE for everything and nothing in particular" (un IDE para todo y para nada en particular), puesto que se puede considerar como únicamente un armazón sobre el que se pueden implementar herramientas de desarrollo para cualquier lenguaje.

5.2.2 NetBeans IDE

NetBeans [Oracle, 2012] es un entorno de desarrollo de código abierto creado en el año 2000 por Sun Microsystems. También dispone de distintos módulos que permiten extender la funcionalidad del entorno.

Aunque está escrito en Java se puede utilizar para desarrollar aplicaciones en cualquier otro lenguaje de programación.

5.2.3 Altova XML Spy 2010

Altova XML Spy es un editor y un entorno de desarrollo para modelar, editar, transformar tecnologías relacionadas con XML.

Ofrece un diseñador gráfico, un generador de código, un conversor de archivos, integración con bases de datos y soporta los formatos XSLT, Xpath, XQuery, WSDL, SOAP, XBRL y Office Open XML (OOXML).

Dispone de una evaluación de prueba de 30 días la cual ha sido utilizada para el desarrollo de este proyecto.

5.2.4 Balsamiq

Balsamiq [Balsamiq, 2008] es una herramienta de creación de mockups que sirve para prototipar o esbozar ideas enfocándose en los factores clave de un proyecto. Es un producto de pago pero dispone de una versión online de prueba, que es la que se ha utilizado para el desarrollo de los prototipos.

Balsamiq incluye los elementos más comunes de una GUI tales como: ventanas, botones, cajas de texto, barras de desplazamiento, etc. Permite salvar los prototipos creados en un fichero XML para editarlos posteriormente y exportar dichos prototipos a una imagen.

5.2.5 Herramientas de modelado UML

Para la realización de los distintos diagramas de diseño se han empleado dos herramientas: *Visual Paradigm for UML* y *ObjectAid UML Explorer (*plugin para el IDE Eclipse).

6. Diseño

Una vez descritos los requisitos que debe satisfacer la aplicación, en este capítulo se define la estructura con la que se ha modelado la solución desarrollada. Dicha estructura viene descrita por la definición de la arquitectura del sistema, que comprende la organización en subsistemas software y la especificación del entorno tecnológico.

En el primer apartado se define el sistema a desarrollar. Posteriormente se ofrece la especificación detallada de los componentes descrito a través del diagrama de clases detallado y el diseño físico de los datos.

6.1 Definición del sistema

La plataforma a desarrollar consistirá en un ASR capaz de incorporar módulos externos en tiempo de ejecución con independencia del sistema. Los módulos se deberán encontrar en el dispositivo de almacenamiento primario, en una ruta accesible por la plataforma.

La plataforma obtendrá la señal de audio a partir de un fichero en formato WAV con modulación PCM.

Al tratarse de una aplicación de escritorio, será necesaria la gestión de la interfaz gráfica de usuario o GUI. Dicha gestión consistirá en proporcionar un entorno visual sencillo que facilitará la comunicación con el sistema operativo, y a su vez posibilitará una interacción amigable entre el usuario y la plataforma.

Será necesaria la existencia de una interfaz que defina claramente que funciones debe implementar el usuario para que el módulo interactúe correctamente con la plataforma Goldfinch. Dicha interfaz consistirá en una colección de funciones, en las que se especificará al usuario que debe hacer el módulo pero en el que no se incluirá una implementación.

Los módulos externos desarrollados por el usuario podrán ser configurados tanto a través de configuraciones predefinidas cargadas desde un fichero en formato XML como a través de la interfaz gráfica de usuario anteriormente mencionada.

La plataforma ofrecerá la posibilidad de calcular estadísticos que definen la precisión y exactitud de la transcripción obtenida. Dichos estadísticos permiten a su vez el conocer el rendimiento de los módulos provistos por el usuario.

6.2 Descomposición en subsistemas

El sistema definido en la sección anterior se ha descompuesto en una colección de componentes. Un componente está definido por las siguientes características:

- Forma parte del sistema.
- Es reemplazable por otro de similares características.
- Conforma y proporciona la realización de un conjunto de interfaces.

Cada componente es una unidad de despliegue independiente que puede ser conectado con otros componentes. En la plataforma Goldfinch existe una única copia de cada componente, cada uno de los cuales realiza una función bien definida. En la definición del sistema un componente supone una unidad autónoma, reemplazable y reutilizable que habitualmente es ejecutable.

En la siguiente tabla se enumeran los componentes que definen el sistema y se describe la función que desempeña:

Componente	Función
Goldfinch App	Subsistema utilizado para arrancar la plataforma.
Goldfinch GUI	Subsistema que gestiona la interfaz gráfica de usuario y la entrada y presentación de datos.
Goldfinch Controller	Subsistema que maneja las peticiones que realiza el usuario.
Goldfinch Configuration	Subsistema utilizado para permitir la configuración de los módulos proporcionados por el usuario.
Goldfinch Java Interfaces	Subsistema que indica al usuario de la plataforma que funciones deben implementar los módulos desarrollados.
Goldfinch Diff	Subsistema que computa las diferencias entre el texto objetivo y la transcripción obtenida.
Goldfinch Commons	Subsistema que contiene las propiedades y funciones comunes a todos los subsistemas de la plataforma.
Goldfinch Modules	Subsistema que contiene la lógica para la gestión de los módulos incorporados por el usuario.
Goldfinch Provider	Subsistema que proporciona los datos que definen la señal de audio contenida en un fichero WAV con modulación PCM.

Tabla 2 - Componentes del sistema y función que desarrollan

En el siguiente diagrama se muestran los componentes citados anteriormente además de las relaciones ó dependencias existentes entre dichos componentes:

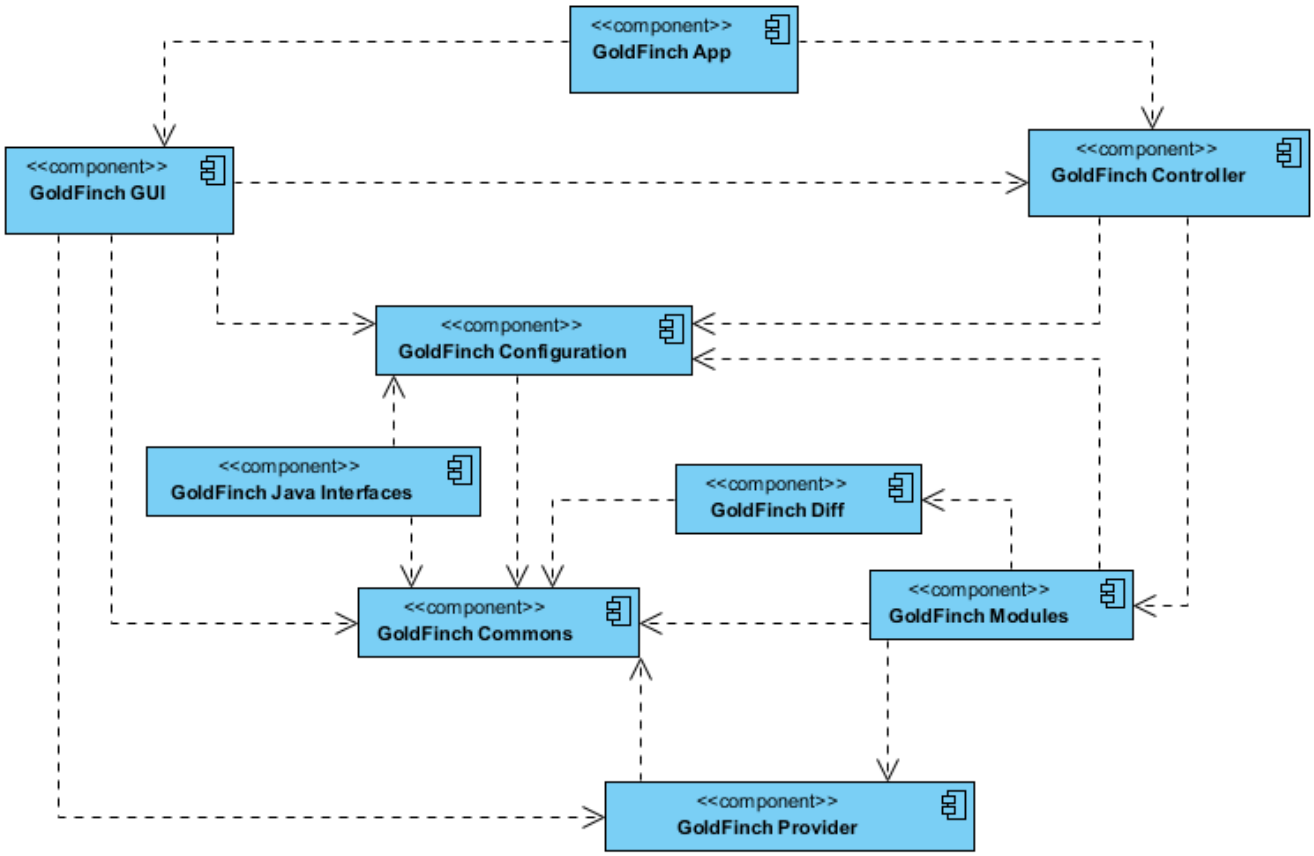


Figura 22 - Diagrama de componentes

Dichas relaciones representan el uso de una o varias de las funcionalidades ofrecidas por un componente por parte de otro.

6.3 Principios de diseño

Una vez definido el sistema y se han citado los componentes que lo forman, en este apartado se citan los principios de diseño sobre los que se basa la arquitectura de la aplicación. Dichos principios son los siguientes:

- **La separación de la lógica y la interfaz de usuario:** Se ha separado la lógica de gestión de los módulos del proyecto que contiene la interfaz empleando el patrón arquitectónico MVC (modelo vista controlador).

- **La separación de lógica y acceso a datos:** El acceso a los datos persistidos y la lógica de la plataforma se encuentran separados en capas diferentes.

- **Independencia entre la plataforma y los módulos desarrollados por el usuario:** Las dependencias entre la plataforma Goldfinch y los módulos de prueba son mínimas, compartiendo únicamente librerías comunes y la interfaz que deben implementar los módulos que serán desarrollados por el usuario de la plataforma.

- **Modularidad:** El diseño se ha realizado teniendo como objetivo fundamental la modularidad de la aplicación de manera que se facilite el desarrollo, las pruebas y el mantenimiento de la misma. La plataforma se ha descompuesto en subsistemas, y cada uno de ellos se ha agrupado en un proyecto distinto en el entorno de desarrollo. Cada uno de estos proyectos es empaquetado en una librería lo que permitiría su utilización en otros proyectos.

6.4 Enfoque de programación

El enfoque de programación que se ha empleado para el diseño de la plataforma se denomina "Programación Orientada a Objetos". Este enfoque está completamente soportado por el lenguaje escogido para el desarrollo de la plataforma (Java) y con él se pueden desarrollar aplicaciones que soporten los principios de diseño definidos en la sección anterior.

La programación orientada a objetos fomenta las siguientes estrategias de desarrollo del software [L. Winblad, et al. 1993]:

- Escribir código reutilizable.
- Escribir código posible de mantener.
- Depurar módulos de código existentes.
- Compartir código con otros.

Los mecanismos de orientación a objetos, en particular la herencia, fomentan la reutilización. En lugar de copiar y modificar módulos, los programadores pueden utilizar librerías de clases conteniendo código comprobado y depurado.

Un sistema orientado a objetos utiliza la abstracción para su eficacia. Las bibliotecas de clases abstractas facilitan el ensamblar las aplicaciones más rápida y fácilmente.

6.5 Diagrama de despliegue

El siguiente diagrama muestra el despliegue de los diferentes componentes del sistema citados en la sección 5.2 de este capítulo. Asimismo ofrece las relaciones entre los componentes lógicos desplegados sobre la arquitectura física utilizada:

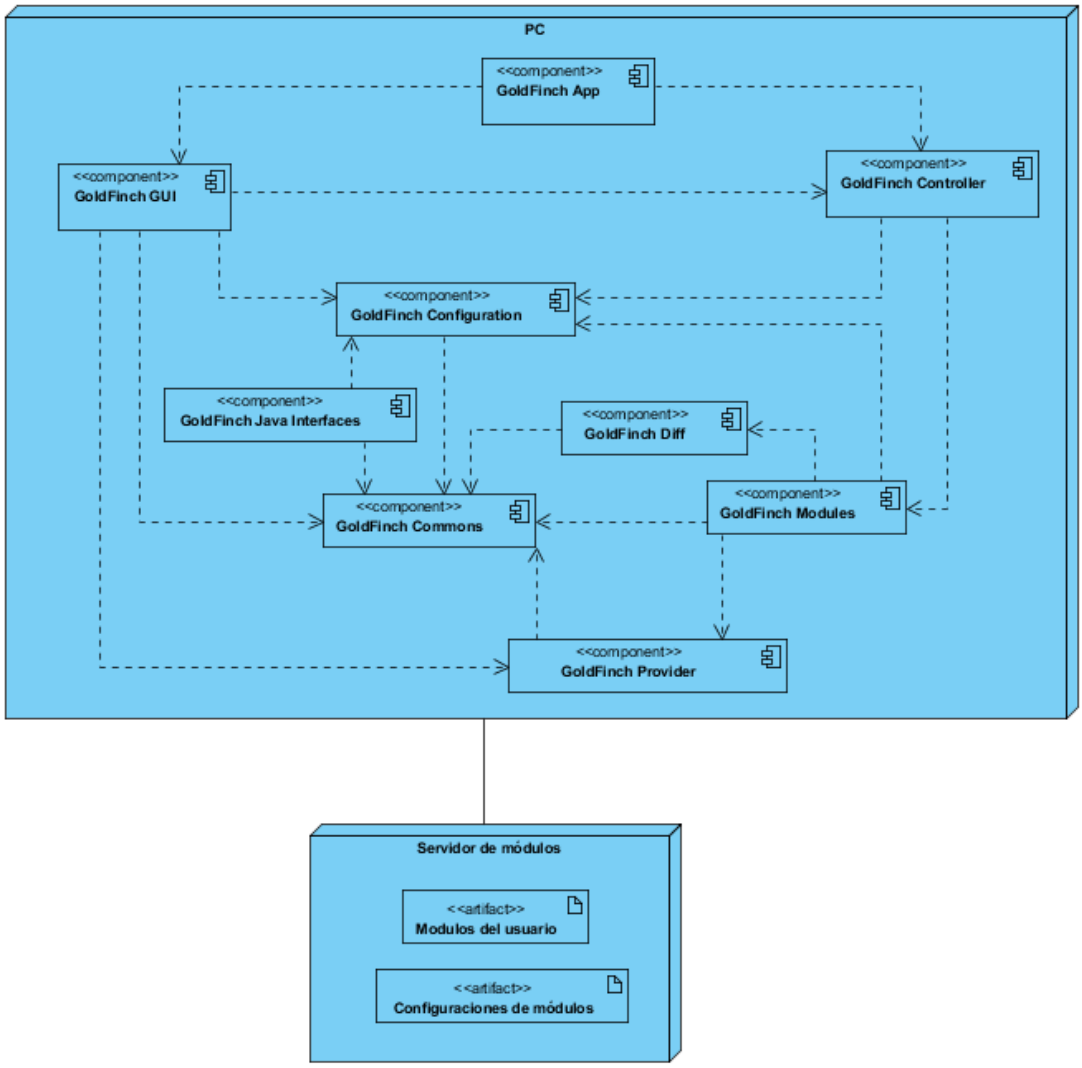


Figura 23 - Diagrama de despliegue

La arquitectura lógica de la máquina utilizada se compone de un ordenador con la máquina virtual de Java instalada (versión mayor o igual a la 1.6) y adicionalmente la posibilidad de usar uno o varios servidores de archivos que contengan tanto los módulos candidatos a ejecutar por el usuario como sus configuraciones.

Las funciones principales de estos elementos son la ejecución de la plataforma utilizando los módulos seleccionados por el usuario dispuestos según la configuración del módulo escogido y que pueden estar almacenados tanto en el PC de ejecución como en un servidor remoto del cual se conoce la ruta y que debe estar habilitado para servir archivos en la misma red en la que se encuentra el entorno de ejecución de la plataforma.

6.6 Descripción de los tipos de módulos de la plataforma

En esta sección se describen los distintos tipos de módulos que gestiona la plataforma.

La comunicación entre módulos se ha resuelto usando cuatro tipos de módulos:

- Módulo antecesor
- Módulo sucesor
- Módulo invocador
- Módulo invocado.

A continuación se ofrece una simplificación de la arquitectura que se ha implementado para facilitar la comunicación entre los distintos módulos que integran la plataforma.

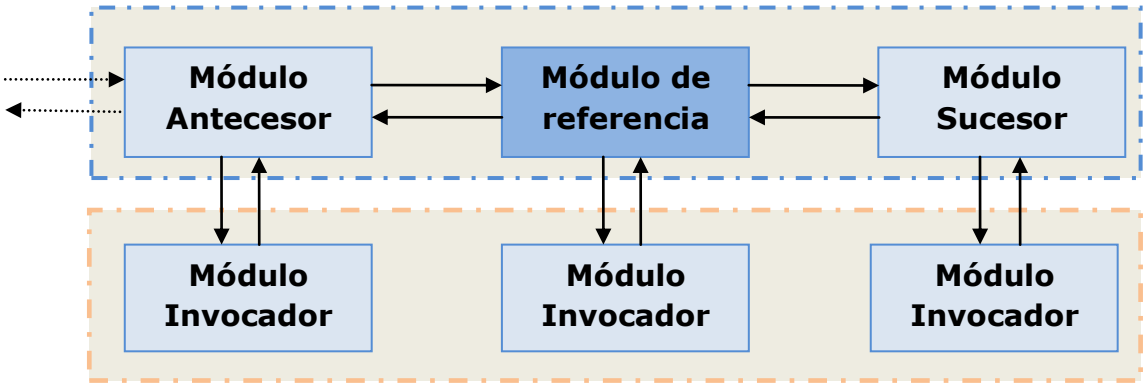


Figura 24 - Comunicación entre los módulos de la plataforma Goldfinch

La Figura 24 muestra las llamadas e invocaciones de un módulo a otro. Se han omitido por claridad las llamadas de los módulos invocadores a los invocados (cargados por los usuarios), pero su esquema de comunicación es idéntico al que mantienen los módulos de reconocimiento con los módulos invocadores.

Todos los módulos basan sus operaciones en las que ofrece un módulo invocador que es el encargado de la comunicación con el módulo externo. Este módulo invocador introduce datos sin procesar y extrae datos procesados según se lo indica el módulo que lo invoca.

Por otro lado los datos una vez procesados fluyen de manera horizontal desde los módulos antecesores a los módulos sucesores de izquierda a derecha. El resultado final hace el camino inverso, volviendo del módulo final al módulo inicial.

Esta abstracción en la plataforma permite la inclusión y eliminación de las relaciones de una manera relativamente sencilla.

6.7 Diseño detallado

En esta sección se presenta la descomposición de los subsistemas que componen la plataforma describiendo detalladamente cada uno de ellos. La definición de las clases y componentes se nombran en inglés para facilitar su correspondencia con los artefactos presentes en el código fuente.

6.7.1 Paquete Goldfinch App

Este subsistema se utiliza únicamente para arrancar la plataforma. El inicio se produce en tres pasos: primero se instancia el controlador, acto seguido se instancia la interfaz de usuario (contenida en el subsistema Goldfinch GUI) y posteriormente se ejecuta el método *startup()* en la GUI que recibe como parámetro el controlador iniciado anteriormente.

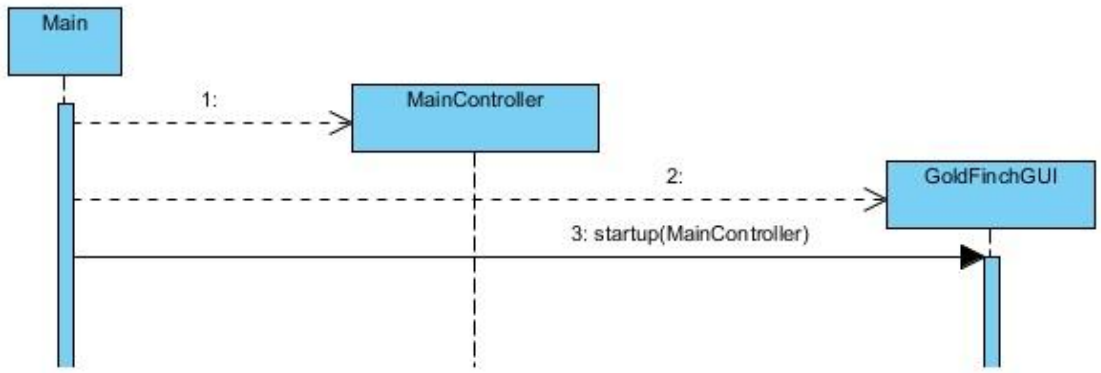


Figura 25 - Diagrama de secuencia de inicio de la plataforma

6.7.2 Paquete Goldfinch Modules

El paquete *Goldfinch Modules* contiene la lógica necesaria para la gestión de los módulos de la aplicación.

A continuación se describen las clases más significativas de dicho subsistema:

La clase abstracta *AbstractModule* posee una referencia circular a sí misma, indicando que cada uno de los módulos tiene un módulo sucesor.

La instanciación y establecimiento de los sucesores se configura desde el paquete que contiene el controlador de la plataforma. La configuración actual de la plataforma se define del siguiente modo:

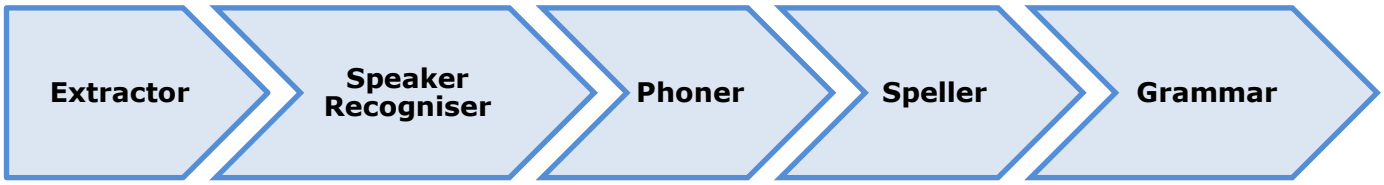


Figura 26 - Configuración de los módulos de la plataforma

- El módulo *Extractor* tiene como sucesor al módulo *Speaker Recogniser* y no posee ningún antecesor (o de otra forma su antecesor es el objeto nulo).
- El módulo *Speaker Recogniser* tiene como sucesor al módulo *Phoner*.
- El módulo *Phoner* tiene como sucesor al módulo *Speller*.
- El módulo *Speller* tiene como sucesor al módulo *Grammar*.

Cada módulo extiende de la clase *AbstractModule*, e incluye los métodos concretos que necesita para efectuar sus operaciones.

Cada módulo delega sus operaciones en las implementaciones que escoja el usuario de la aplicación. Para ello se han creado unas clases denominadas invocadores que son las encargadas de realizar las llamadas a las implementaciones escogidas por el usuario. La interfaz común a todos los invocadores es la interfaz *Invoker*.

La interfaz *Invoker* contiene la definición de tres métodos:

- **getModuleType**: devuelve el tipo de módulo.
- **setConfiguration**: establece la configuración en el módulo proporcionado por el usuario.
- **getDefaultConfigurationPath**: obtiene la ruta del archivo de configuración por defecto para el módulo del usuario.

Cada uno de las clases invocadoras tiene dos métodos para cada uno de los módulos:

- Un método *push* que introduce datos sin procesar por el módulo.
- Un método *pop* que extrae datos procesados por el módulo.

Dichos métodos están definidos en la interfaz correspondiente. La interfaz *Invoker* es extendida a su vez por tantas interfaces como módulos existen. Cada una define el tipo de datos que envía al módulo del usuario y el tipo de datos que recibe del este.

A continuación se exponen las signaturas de los métodos de las interfaces que definen a los *Invokers*:

Nombre de la clase	Método	Tipo de dato que recibe	Tipo de dato que devuelve
ExtractorInvoker	Push	Vector de números en precisión doble	-
	Pop	-	Vector de números en precisión doble
SpeakerRecogniserInvoker	Push	Vector de números en precisión doble	-
	Pop	-	Cadena alfanumérica.
PhonerInvoker	Push	Vector de números en precisión doble	-
	Pop	-	Lista de objetos *Result*
SpellerInvoker	Push	Lista de objetos *Result*	-
	Pop	-	Lista de objetos *Result*
GrammarInvoker	Push	Lista de objetos *Result*	-
	Pop	-	Lista de objetos *Result*

Tabla 3 - Signaturas de las interfaces de los Invokers

Un objeto *Result* incluye una cadena alfanumérica para definir un identificador y un número en precisión doble para indicar la probabilidad de aparición de ese resultado.

6.7.3 Paquete Goldfinch Commons

El paquete Goldfinch Commons contiene las propiedades y las funciones que son comunes al resto de paquetes de la plataforma.

- Los valores modificables se encarga de gestionarlos la clase *Configuration* que implementa el patrón *Singleton*. Dicha clase lee un conjunto de pares clave-valor de un archivo llamado *Goldfinch-conf.properties* almacenado en este mismo paquete.

- Los valores no modificables se encuentran en la clase *Constants*, y son almacenados directamente en variables precedidos del modificador *final* que indica que es una constante.

A su vez incluye clases de útiles con métodos estáticos que pueden ser invocados desde cualquier paquete que haga referencia a este.

Al objeto de permitir configurar la plataforma de una manera sencilla, se ha proporcionado un mecanismo de gestión de configuración que lee los valores de un archivo de extensión *.properties* y los almacena en memoria secundaria de modo que pueda ser cargado por cualquier proyecto que integre la plataforma y tenga dependencia con el proyecto encargado de gestionar la configuración.

La extensión *.properties* se utiliza en ficheros usados para almacenar parámetros de la aplicación en aplicaciones Java. Dichos archivos almacenan texto plano cuyos parámetros tienen un formato predeterminado. El archivo de configuración para la plataforma tiene el nombre *goldfinch-conf.properties* y se carga al arrancar la aplicación.

Cada parámetro es almacenado en un par de cadenas de texto. La primera cadena de texto contiene la clave unívoca del parámetro. La segunda cadena de texto contiene el valor que toma dicho parámetro.

Se admiten distintos formatos para el almacenamiento de la clave y el valor, siendo equivalente el uso de *clave=valor, clave = valor, clave:valor y clave valor*. Se debe utilizar una línea distinta para cada par clave valor.

El archivo de configuración admite comentarios, cuyas líneas deben ir precedidos de los símbolos ! *ó #.*

Tanto la ruta donde se encuentra el archivo como los valores que pueden almacenarse son totalmente configurables, y requieren cambios tanto en el propio archivo de configuración como en la clase que gestiona dicho archivo. Las claves se almacenan en constantes estáticas para que el acceso a los valores sea común por todos los proyectos que integran la plataforma. De este modo en un IDE avanzado es sencillo conocer qué valores se pueden obtener con el modo de sugerencia. También previene el que se cometan errores al escribir las claves para obtener los valores de configuración.

Actualmente el archivo contiene valores de configuración como el canal a utilizar por defecto si el archivo contiene audio en estéreo, el número de muestras que recibe el extractor, la resolución del gráfico mostrado en la interfaz o los nombres y el paquete por defecto utilizado en los módulos proporcionados por el usuario.

A continuación se muestra un ejemplo de un archivo de configuración:

```
## stereo default channel number
stereo.default.channel.number = 1

## samples to extractor (number)
samples.extractor.number = 1

## graph resolution (for establish the number of samples to display in
the graph)
gui.graph.resolution = 0.1

## modules class names
module.jar.extractor.classname = ExtractorExample
module.jar.recogniser.classname = RecogniserExample
module.jar.phoner.classname = PhonerExample
module.jar.speller.classname = SpellerExample
module.jar.grammar.classname = GrammarExample

## modules packages names
module.jar.extractor.packagename = modules
module.jar.recogniser.packagename = modules
module.jar.phoner.packagename = modules
module.jar.speller.packagename = modules
module.jar.grammar.packagename = modules
```

6.7.4 Paquete Goldfinch Diff

Las operaciones que se efectúan para comparar el texto de la transcripción objetivo y el resultado obtenido se han englobado dentro del paquete *Goldfinch Diff*.

La clase que implementa la interfaz *DiffManager* recibe los dos textos, los alinea, computa las diferencias y devuelve un objeto *DiffResume* con los resultados de la comparación.

Se incluyen dos implementaciones de la interfaz *DiffManager*. La clase *SphinxDiffManagerImpl* es la utilizada por CMU Sphinx-4. La clase

RomanowDiffManagerImpl incluye una implementación libre encontrada en el repositorio de código público llamado *GitHub*.

Después de analizar dos archivos de texto se obtendrá un objeto *DiffResume* que aportará información sobre palabras añadidas, palabras eliminadas, palabras modificadas, palabras iguales y distancia de Levenshtein entre los dos textos propuestos.

6.7.5 Paquete Goldfinch Provider

El paquete *Goldfinch Provider* contiene la lógica para la obtención de una colección de bytes a partir de archivos que representan archivos de audio.

Las propiedades de un archivo de audio son descritas por la clase *AudioProperties*.

La interfaz principal de este paquete (denominada *ProviderManager*) proporciona los siguientes métodos:

- **play:** Reproduce la pista de audio a partir de los objetos que la representan.
- **getLine:** Obtiene una línea a partir de un objeto *AudioFormat.*
- **getAudioProperties:** Obtiene un objeto *AudioProperties* que simboliza las características comunes (frecuencia, nombre del archivo, frame rate, bytes por frame, etc.) de un archivo de audio.
- **getAudioInputStream:** Obtiene un stream de audio a partir de la ruta del archivo de audio proporcionada.
- **obtainArrayStream:** Obtiene un array de bytes a partir del stream de audio proporcionado.

Este paquete contiene además una clase de útiles denominada *SampleUtils*. Dicha clase obtiene valores en coma flotante normalizados a partir de un array de bytes.

Organización de muestras en una matriz de bytes

Para normalizar las muestras es necesario identificar como se organizan las muestras en su representación en una matriz de bytes. Esto permitirá efectuar las operaciones de bit pertinentes para que las muestras estén representadas en el intervalo **[-1, 1]**.

Las muestras se diferencian en: si son Big Endian/Little Endian, en si son mono o estéreo, en el número de canales y en el número de bytes que emplean.

Endianness

La mayoría de los ordenadores tienen su memoria organizada en unidades de 8 bits llamadas bytes. Dichos bytes están identificados por números que comienzan en el cero. Si se necesitan almacenar más de 8 bits existirán dos grupos (del bit 0 al bit 7 y del bit 8 al bit 15). Algunos procesadores almacenan el primer grupo en el byte de menor dirección, y el segundo grupo en el byte de mayor dirección. Este esquema se denomina Little Endian. En cambio si la organización es al revés, el esquema utilizado se denomina Big Endian.

Para Java Sound dicha organización de bytes en la memoria solo tiene sentido si las muestras tienen una longitud mayor a 8 bits. Las implementaciones de las funciones principales de Java Sound pueden utilizar ambos endianness de manera indistinta.

Muestras con signo / muestras sin signo

En PCM los valores de las muestras son representadas por números enteros. Estos números enteros pueden tener signo (estableciendo el valor central entre el máximo y el mínimo valor en 0) o no tener signo, en cuyo caso el menor valor será el 0.

La tabla 4 describe las representaciones que han sido implementadas en este proyecto tomando como referencia matrices de 4 bytes de longitud.

bits	Mono/estéreo	Canales	Little Endian / Big Endian	Byte 0	Byte 1	Byte 2	Byte 3
8	Mono	1	Indiferente	Muestra completa	Muestra completa	Muestra completa	Muestra completa
8	Estéreo	2	Indiferente	1ª Trama, muestra izquierda	2ª Trama, muestra derecha	2ª Trama, muestra izquierda	2ª Trama, muestra derecha
16	Mono	1	Little Endian	1ª muestra, byte menor	1ª muestra, byte mayor	2ª muestra, byte menor	2ª muestra, byte mayor
16	Mono	1	Big Endian	1ª muestra, byte mayor	1ª muestra, byte menor	2ª muestra, byte mayor	2ª muestra, byte menor
16	Estéreo	2	Little Endian	1ª Trama, byte menor	2ª Trama, byte mayor	2ª Trama, byte menor	2ª Trama, byte mayor
16	Estéreo	2	Big Endian	1ª Trama, byte mayor	2ª Trama, byte menor	2ª Trama, byte mayor	2ª Trama, byte menor
32	Mono	1	Little Endian	1ª Trama, 1ª muestra, 4ª byte (bits 24-31)	1ª Trama, 1ª muestra, 3º byte (bits 16-23)	1ª Trama, 1ª muestra, 2º byte (bits 8-15)	1ª Trama, 1ª muestra, 1º byte (bits 0-7)
32	Mono	1	Big Endian	1ª Trama, 1ª muestra, 1º byte (bits 0-7)	1ª Trama, 1ª muestra, 2º byte (bits 8-15)	1ª Trama, 1ª muestra, 3º byte (bits 16-23)	1ª Trama, 1ª muestra, 4ª byte (bits 24-31)

Tabla 4 - Organización de muestras en matrices de bytes

Tamaño de la muestra	Endianness/Signedness	Código
8 bits	Signed	`buffer[offset] / 128.0;`
8 bits	Unsigned	`((buffer[offset] & 0xFF) - 128)/ 128.0;`
16 bits	Little Endian	`((buffer[offset + 0] & 0xFF) | (buffer[offset + 1] << 8)) / 32768.0;`
16 bits	Big Endian	`((buffer[offset + 0] << 8) | (buffer[offset + 1] & 0xFF)) / 32768.0;`
24 bits	Little Endian	`((buffer[offset + 0] & 0xFF) | ((buffer[offset + 1] & 0xFF) << 8) | (buffer[offset + 2] << 16)) / 8388606.0;`
24 bits	Big Endian	`((buffer[offset + 0] & 0xFF) | ((buffer[offset + 1] & 0xFF) << 8) | (buffer[offset + 2] << 16)) / 8388606.0;`
32 bits	Little Endian	`((buffer[offset + 0] & 0xFF) | ((buffer[offset + 1] & 0xFF) << 8) | ((buffer[offset + 2] & 0xFF) << 16) | (buffer[offset + 3] << 24) ) / 2147483648.0;`
32 bits	Big Endian	`((buffer[offset + 0] << 24) | ((buffer[offset + 1] & 0xFF) << 16) | ((buffer[offset + 2] & 0xFF) << 8) | (buffer[offset + 3] & 0xFF) )/ 2147483648.0;`

Tabla 5 - Operaciones para normalización de muestras

6.7.6 Paquete Goldfinch Configuration

El paquete *Goldfinch Configuration* contiene las clases necesarias para leer la configuración contenida en archivos XML para los módulos proporcionados por el usuario.

Para realizar su cometido se basa en la librería de Java JAXB, por lo que se han incluido las clases que se generan invocando a JAXB vía línea de comandos. Dichas clases de acceso a datos se transforman en clases pertenecientes al dominio de la plataforma.

El método obtainConfiguration de la clase *ConfigurationManager* obtiene la ruta del archivo configuración en XML, y devuelve un objeto *ConfigurationBO* que contiene toda la configuración. Si ocurre algún error se propaga la excepción descrita por la clase *ConfigurationException*.

Para permitir la validación de los archivos en XML también se incluye el esquema en el archivo *Configuration.xsd* en este paquete.

Almacenamiento de configuración de módulos del usuario

La configuración para los módulos que proporciona el usuario debe ser cargada y gestionada por la plataforma con el objeto de permitir su modificación y posterior validación. La plataforma una vez obtiene los valores de configuración validados los incorpora en cada uno de los módulos proporcionados por el usuario, donde se debe gestionar la recepción de dichos parámetros.

Se ha escogido XML para el almacenamiento de los valores de configuración de los módulos cargados por el usuario. Las razones que justifican esta decisión son las siguientes:

- Se pueden modificar los valores de configuración en un futuro con nuevas etiquetas de manera sencilla, con pocos cambios en el código ya desarrollado.
- Existen numerosas librerías y documentación para el procesamiento de XML en numerosos lenguajes de programación puesto que es un formato muy extendido.
- Es sencillo entender, crear y editar la estructura del archivo de configuración, puesto que XML está basado en texto.
- Se pueden definir archivos XSD para describir formalmente que elementos debe contener un documento XML. Dicha descripción puede ser usada para verificar que cada elemento del contenido del documento se adhiere a la definida en el archivo XSD. Los archivos XSD además se definen utilizando XML.

A continuación se describen los principales elementos de un archivo de configuración:

Elemento Configuration

El archivo XML de configuración posee un elemento raíz denominado <Configuration>. Dentro de dicho elemento se definen los atributos xsi:noNamespaceSchemaLocation y xmlns:xsi para describir la localización del archivo XSD de definición del esquema y la definición del namespace respectivamente. El elemento Configuration debe contener uno o más elementos ConfigurableValue.

Elemento ConfigurableValue

El elemento <ConfigurableValue> define un valor configurable. Dicho valor Configurable puede ser de cuatro tipos distintos (Multiple, IntegerValue, Generic y DoubleValue).

Dichos tipos se describen en la tabla siguiente:

Nombre del elemento	Descripción
Multiple	Describe un elemento de configuración cuyo valor puede elegirse entre una variedad de opciones.
IntegerValue	Elemento utilizado para configurar opciones que se definen utilizando números enteros.
Generic	Elemento utilizado para configurar opciones que se definen utilizando una cadena alfanumérica.
DoubleValue	Elemento utilizado para configurar opciones que se definen utilizando números en precisión doble.

Tabla 6 - Subelementos que extienden el elemento ConfigurableValue

Elemento Multiple

El elemento <Multiple> constituye un valor configurable formado por una lista de elementos Generic. La etiqueta que lo define contiene los atributos name y isExclusive que sirven para describir el nombre del valor configurable y si su valor puede ser único o puede ser definido por una colección de opciones.

Elemento IntegerValue

El elemento <IntegerValue> constituye un valor configurable formado por un valor entero. El attributo name define el nombre del valor configurable mientras que los atributos maxValue y minValue sirven para describir el rango de valores sobre el que puede estar comprendido el elemento.

Elemento Generic

El elemento <Generic> define los valores configurables que pueden ser descritos por una cadena alfanumérica. Contiene el atributo nombre que define el nombre del valor configurable.

Elemento DoubleValue

El elemento <DoubleValue> tiene la misma estructura que el elemento <IntegerValue> pero en lugar de almacenar valores enteros, almacena valores con precisión doble. El valor máximo y mínimo por tanto también deberán expresarse utilizando un número con precisión doble.

Correspondencia entre valores configurables y la interfaz

Para poder seleccionar los valores de configuración deseados, se ha implementado un formulario, el cual está formado por instancias de clases de Java Swing. La correspondencia entre las clases utilizadas y los valores de configuración viene definida en la siguiente tabla:

Valor configurable	Clase Java Swing
Multiple	JRadioButton si el atributo isExclusive es verdadero.
	JCheckBox si el atributo isExclusive es falso.
Generic	JTextField
IntegerValue	JTextField
DoubleValue	JTextField

Tabla 7 - Correspondencia entre elementos de configuración y clases de Java Swing

6.7.7 Paquete Goldfinch Java Interfaces

El paquete *Goldfinch Java Interfaces* contiene las interfaces que deben implementar los módulos proporcionados por los usuarios. Dichas interfaces contienen todos los métodos necesarios para la gestión de dichos módulos.

En este paquete no se incluye ninguna implementación de dichas interfaces pues las implementaciones de las interfaces contenidas en este paquete serán las clases que proporcionará el usuario.

Dicho paquete constituye la única dependencia entre los módulos proporcionados por el usuario y la plataforma.

6.7.8 Paquete Goldfinch Controller

Este paquete implementa el controlador del patrón arquitectónico MVC, es decir, únicamente gestiona las peticiones que realiza la vista (el usuario) y que usualmente suponen peticiones al modelo.

Las peticiones al modelo las realiza a través de la fachada *EnvironmentFacade* definida en el paquete Goldfinch Module.

6.7.9 Paquete Goldfinch GUI

En este paquete se definen los componentes que definen la vista (metáfora del patrón MVC) y el comportamiento de los eventos invocados por el usuario.

Por otro lado contiene las clases que gestionan la validación de la configuración proporcionada por el usuario.

La lectura del archivo de audio que se hace a través del *ProviderManager* está contenida dentro de un hilo de ejecución (thread) para que su ejecución no afecte a la interacción del usuario con la interfaz.

Asimismo contiene los recursos estáticos (imágenes, iconos y archivos de internacionalización).

A continuación se expone el algoritmo de Ramer-Douglas-Peucker utilizado en el paquete GUI para mejorar el rendimiento en la visualización de la gráfica de amplitud de la señal y disminuir el número de puntos a dibujar.

Algoritmo de Ramer-Douglas-Peucker

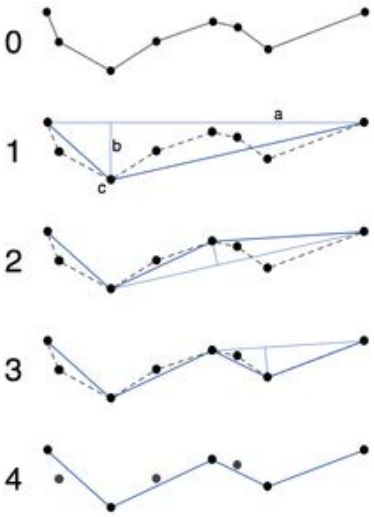
Al objeto de reducir el número de puntos a graficar en la pantalla principal de la plataforma Goldfinch y disminuir así el tiempo de visualización del mismo, se ha implementado el algoritmo de Ramer-Douglas-Peucker.

La forma inicial de este algoritmo fue sugerido de manera independiente por Urs Ramer en 1972 y en 1973 por David Douglas y Thomas Peucker e igualmente por otros en la década posterior [Saalfeld, 1999].

El propósito fundamental es, dada una curva compuesta de segmentos lineales, el encontrar una curva similar con menos puntos. La curva simplificada consiste en un subgrupo de puntos que definen la curva original.

El algoritmo es usado en el procesamiento de gráficos vectoriales y generalización cartográfica.

Explicación del algoritmo



La curva inicial es un grupo ordenado de puntos o líneas y la distancia de dimensión épsilon > 0.

El algoritmo divide la línea recursivamente. Inicialmente está formada por todos los puntos entre el punto inicial y el punto final. Automáticamente se marca el punto inicial y el punto final a conservar. A continuación se marca el punto más alejado del segmento de línea de la curva de aproximación entre los puntos iniciales. Si el punto está más cerca que épsilon al segmento y no hay puntos marcados que puedan ser descartados, entonces la curva será peor que épsilon.

Si el punto más alejado del segmento de línea es mayor que épsilon, ese punto se debe mantener.

El algoritmo se llama recursivamente con el primer punto y el peor punto y posteriormente con el peor punto y el último punto.

Pseudocódigo

A continuación se presenta en pseudocódigo como se ha implementado el algoritmo de Ramer-Douglas-Peucker:

```
function DouglasPeucker(PointList[], epsilon)
 //Find the point with the maximum distance
 dmax = 0
 index = 0

 for i = 2 to (length(PointList) - 1)
  d = PerpendicularDistance(PointList[i], Line(PointList[1], PointList[end]))
  if d > dmax
   index = i
   dmax = d
  end
 end

 //If max distance is greater than epsilon, recursively simplify
 if dmax >= epsilon
  //Recursive call
  recResults1[] = DouglasPeucker(PointList[1...index], epsilon)
  recResults2[] = DouglasPeucker(PointList[index...end], epsilon)

  // Build the result list
  ResultList[] = {recResults1[1...end-1] recResults2[1...end]}
 else
  ResultList[] = {PointList[1], PointList[end]}
 end

 //Return the result
 return ResultList[]
end
```

6.8 Patrones arquitectónicos

Los patrones arquitectónicos expresan modelos para la organización estructural de los sistemas de software, proporcionando un conjunto de subsistemas predefinidos, especificando sus responsabilidades e incluyendo reglas y guías para organizar las relaciones entre ellos.

A continuación se describe el patrón arquitectónico Modelo Vista Controlador que define en gran medida la arquitectura de la plataforma.

6.8.1 Modelo-Vista-Controlador (MVC)

Con el objetivo de reducir las dependencias entre la interfaz gráfica del usuario, la gestión de las peticiones del usuario y la lógica de la aplicación, se ha implementado el patrón Modelo-Vista-Controlador.

Dicha implementación favorece la modularidad de la plataforma, definida anteriormente como uno de los principios de diseño que se han seguido en el desarrollo de la plataforma. De este modo es posible sustituir cualquiera de estos elementos de manera que dichos cambios no influyan en el resto. Por ejemplo sería posible sustituir la vista actualmente basada en una interfaz gráfica de usuario compuesta por ventanas por una vista basada en línea de comandos sin necesidad de realizar modificaciones en el modelo o el controlador.

El Modelo-Vista-Controlador es un patrón arquitectónico que ofrece la separación de los datos de la aplicación, la interfaz de usuario y la lógica de negocio en tres componentes o capas distintas que son definidas como metáforas [E. Krasner, et al. 1988]. Dichas metáforas son la Vista, el Controlador y el Modelo:

- **Vista:** Muestra la información del modelo al usuario. Recibe las acciones del usuario que delega en el controlador para que las gestione.
- **Controlador:** Gestiona las peticiones que realiza el usuario que normalmente suponen peticiones al modelo.
- **Modelo:** Lo componen los datos y las reglas de negocio. Este componente es el encargado de acceder a la capa de almacenamiento de datos y definir las reglas de negocio.

El ciclo de interacción estándar en la metáfora del Modelo-Vista-Controlador supone que el usuario realizar una acción en la aplicación y el controlador notifica al modelo el cambio. El modelo lleva a cabo las operaciones prescritas (es posible que deba cambiar su estado) y notifica al controlador y la vista que ha cambiado, indicándoles incluso la naturaleza del cambio. La vista puede consultar el modelo acerca de su nuevo estado y actualizar su aspecto si fuera necesario.

En el siguiente diagrama se han superpuesto cada una de las capas correspondientes al patrón arquitectónico MVC englobando a cada uno de los componentes que los representan, y los flujos de comunicación entre dichas capas:

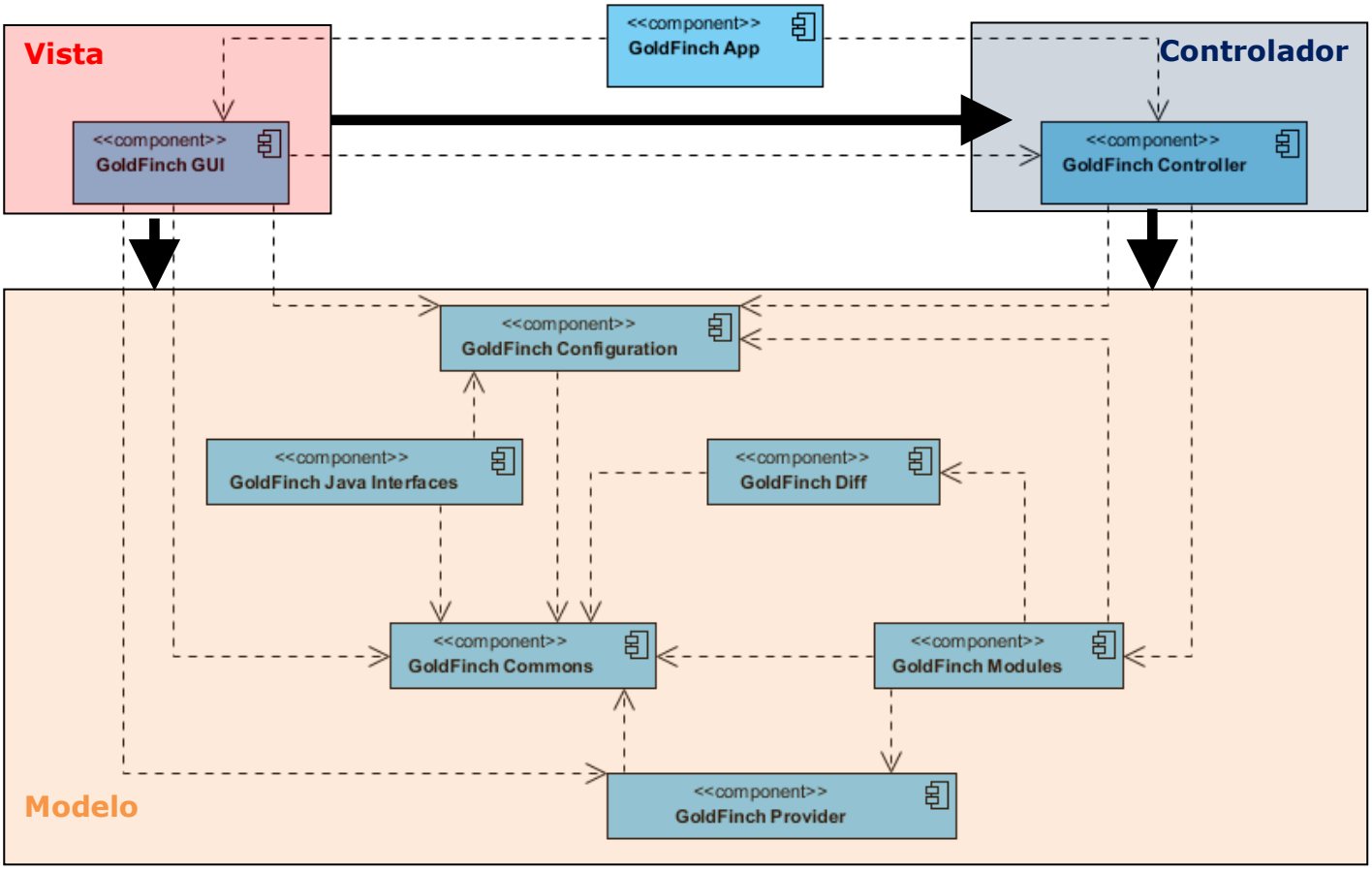


Figura 27 - Diagrama de componentes

6.9 Patrones de diseño

Los patrones de diseño expresan esquemas para definir estructuras con las que construir sistemas software. Estos patrones resuelven un problema de diseño general en un contexto particular. También permiten reutilizar buenos diseños y arquitecturas, y favorecen la reutilización de los sistemas [Gamma, et al. 2006].

Un patrón de diseño describe un problema que ocurre en repetidas ocasiones así como la solución a dicho problema, de tal modo que se pueda reutilizar la solución. Cada patrón por si mismo define, explica y evalúa un diseño importante y recurrente en los sistemas orientados a objetos.

En los siguientes apartados se detallan los patrones de diseño que se han utilizado en la construcción de la plataforma.

6.9.1 Patrón Singleton

El patrón Singleton está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

Se ha utilizado dicho patrón para la gestión de las propiedades configurables, como por ejemplo la longitud de una trama en bits.

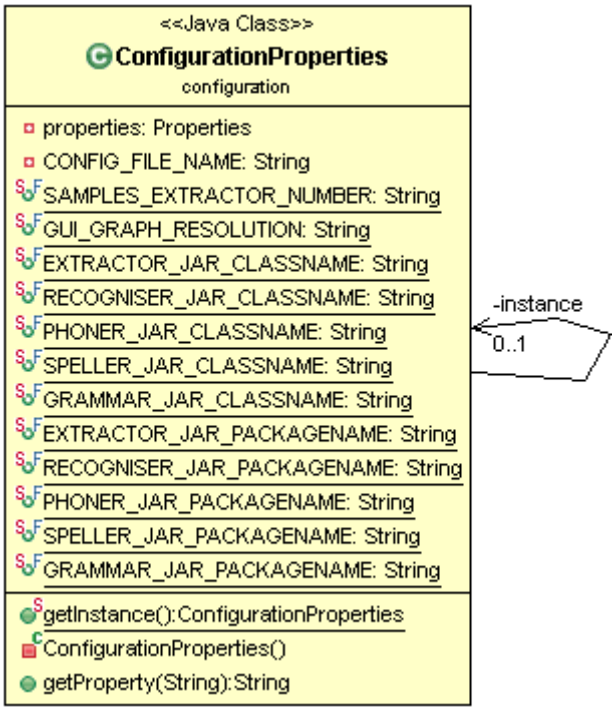


Figura 28 - Diagrama de la implementación del patrón Singleton

Dichas propiedades pueden ser leídas desde cualquier proyecto y están almacenadas en un único fichero con extensión "*.properties*". La primera vez que se invoca al gestor de configuración carga los todos pares clave-valor en un objeto en memoria primaria para poder ser obtenidos posteriormente sin la necesidad de acceder a memoria secundaria.

El patrón garantiza que la gestión de dicho archivo se realizará desde un único lugar, en una única instancia y controla que sólo se realizará un acceso a memoria secundaria para leer la configuración.

6.9.2 Patrón Memento

El patrón de diseño Memento es un patrón de comportamiento que sirve para almacenar el estado de un objeto de forma que pueda ser restaurado posteriormente. La característica principal del patrón memento es que no viola el encapsulamiento.

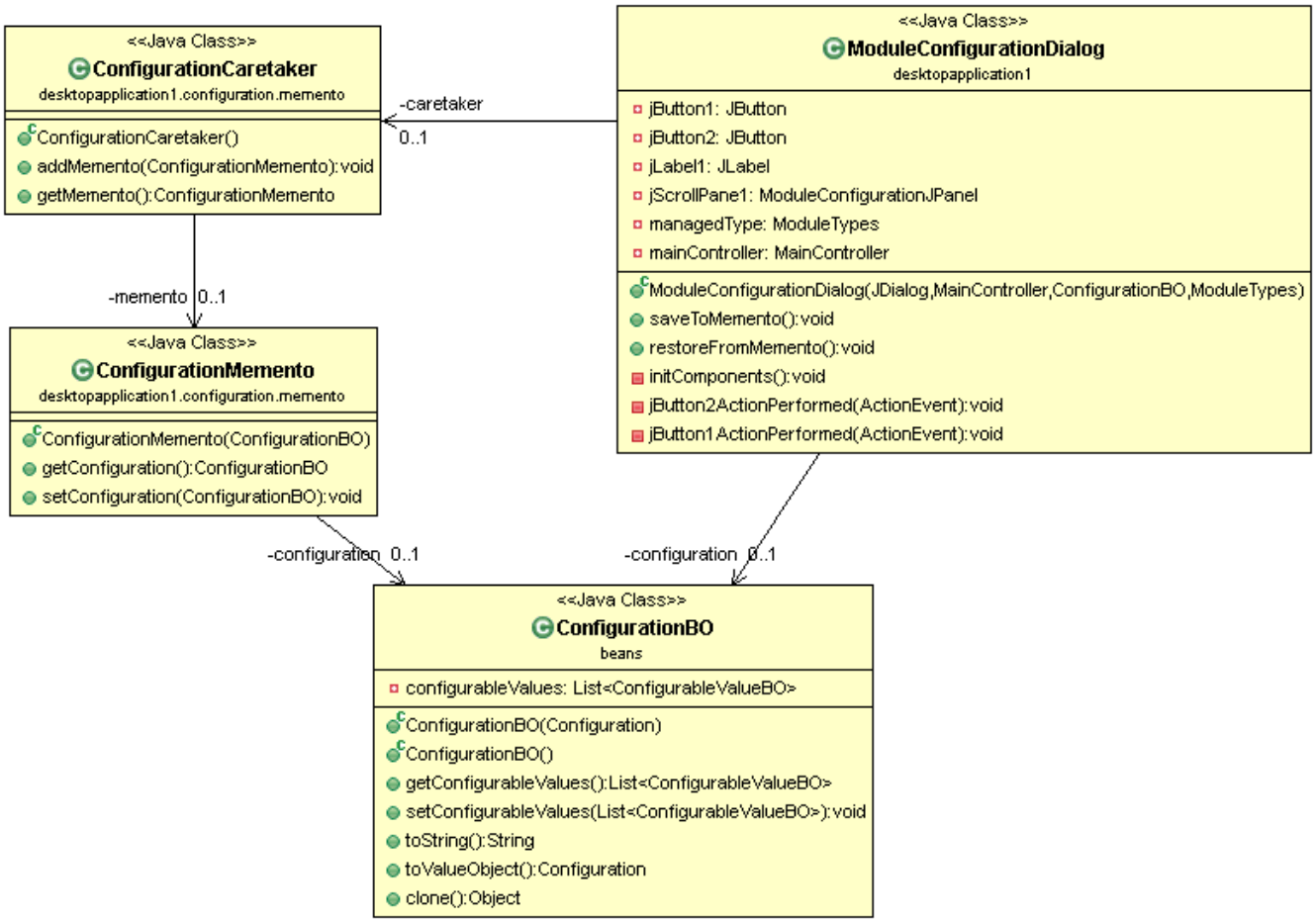


Figura 29 - Diagrama de la implementación del patrón memento

El patrón memento ha sido utilizado para preservar la configuración previa a su edición. La motivación es que el uso de Beans Binding (JSR295) implica que la transferencia del valor modificado en el formulario y el valor almacenado en el objeto de configuración sea inmediata.

Como se puede observar en la Figura 29 la clase *ConfigurationMemento* permite almacenar el estado que en este momento tenga una instancia de la clase *ConfigurationBO*. La clase *ConfigurationCaretaker* es la responsable de almacenar un objeto de tipo *ConfigurationMemento* pero no opera ni examina su contenido.

A su vez la clase *ModuleConfigurationDialog* que representa la ventana de diálogo donde se editan las configuraciones es capaz de almacenar y restaurar su estado en la configuración en la clase *ConfigurationMemento*.

6.9.3 Patrón Factory Method

Los patrones de creación como el patrón Factory Method permiten que el sistema sea independiente de cómo se crean, componen o representan sus objetos. Dichos patrones encapsulan el conocimiento sobre las clases concretas que se van a utilizar, ocultando la forma de crear objetos y permitiendo cierta independencia acerca de:

- Qué objeto se crea.
- Quién crea dicho objeto.
- Cómo se crea ese objeto.
- Cuándo se crea este objeto.

Se ha utilizado este patrón para la creación de los distintos módulos de la plataforma.

En la Figura 31 se muestra la implementación del patrón Factory Method utilizado para crear a los invocadores. Se han eliminado las clases abstractas e interfaces que implementan los módulos por claridad.

Como se puede observar todos los módulos se crean desde la factoría abstracta llamada *ModuleFactory*. Su implementación denominada *ModuleFactoryImpl* permite extensibilidad, pues se delega la creación de los objetos en operaciones separadas de forma que el usuario podría cambiar la implementación de las operaciones de creación de manera sencilla.

El objeto que invoca a la creación de los módulos es la fachada denominada *EnvironmentFacadeImpl* que será explicada en la siguiente sección.

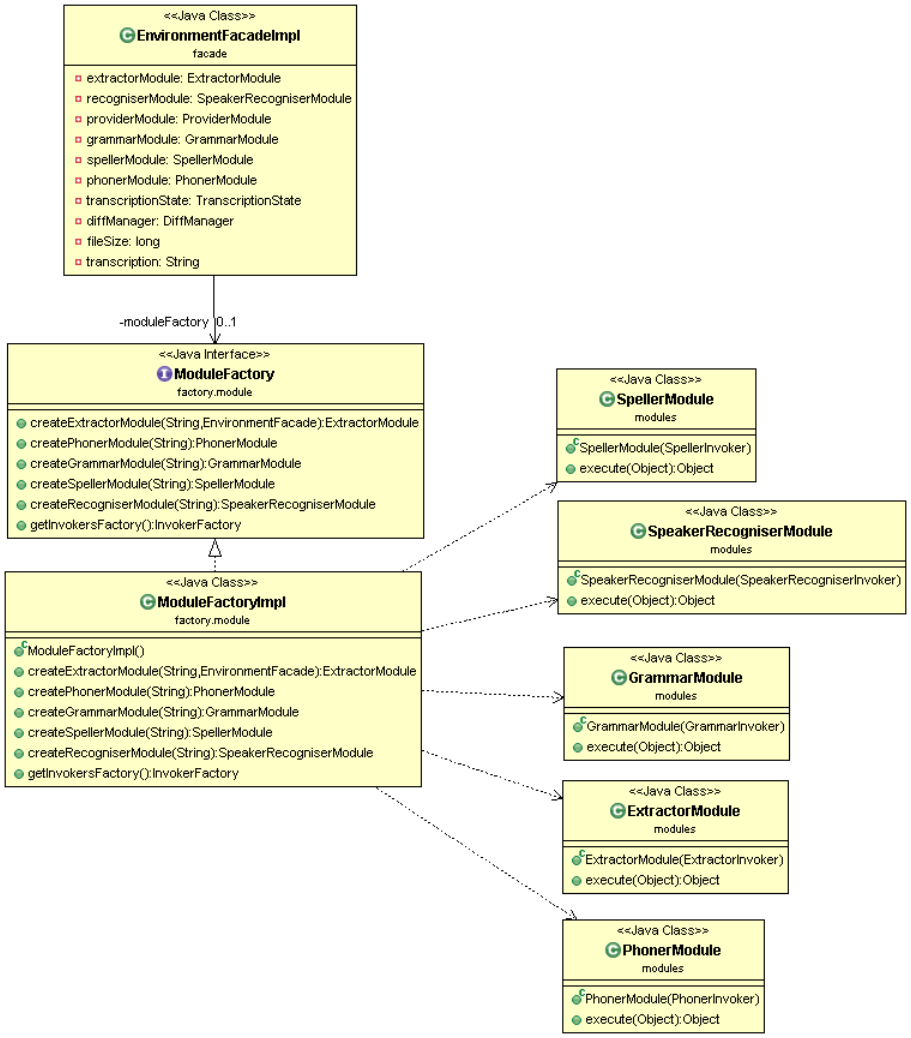


Figura 30 - Diagrama de la implementación del patrón Factory Method para módulos

De manera análoga a la factoría de creación de módulos existe una factoría de creación de invocadores que permite la creación de tantos invocadores como módulos existen.

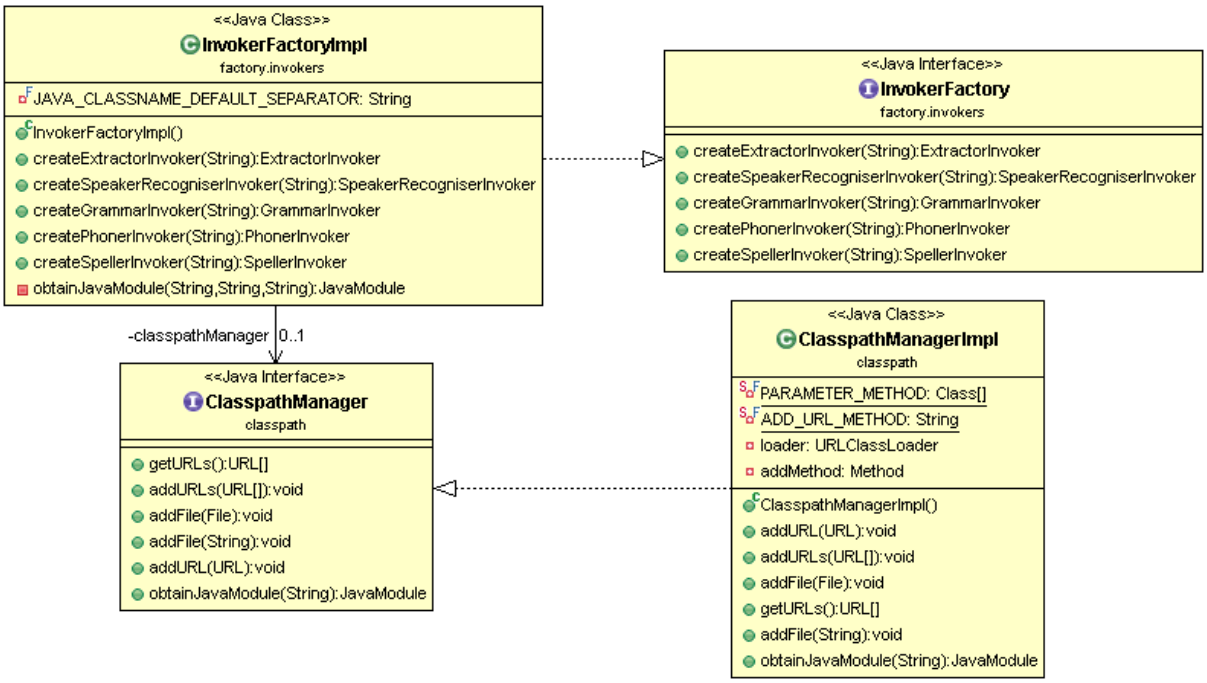


Figura 31 - Diagrama de implementación del patrón Factory Method para invocadores

Dicha factoría basa su funcionamiento en una clase llamada *ClasspathManager* que gestiona que módulos han sido cargados en el Classpath de Java.

6.9.4 Patrón Facade

El patrón *Facade* proporciona una interfaz unificada para un conjunto de interfaces de un sistema. Dicho patrón define una interfaz de alto nivel que facilita el uso del subsistema que encapsula.

Este patrón de diseño se clasifica dentro de los denominados patrones estructurales.

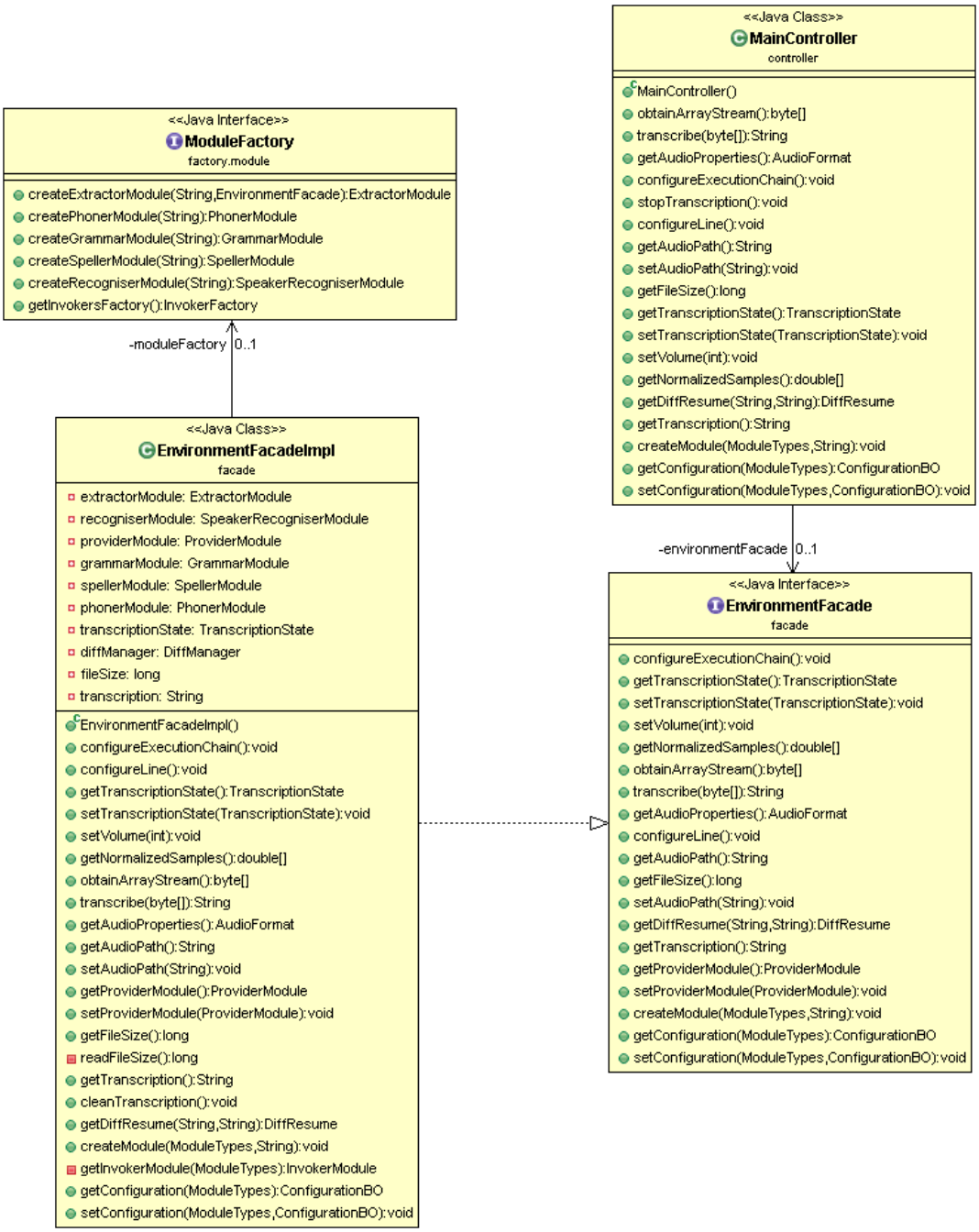


Figura 32 - Diagrama de la implementación del patrón Facade

El patrón Facade está descrito por la interfaz *EnvironmentFacade* que es implementada por la clase *EnvironmentFacadeImpl*. Como anteriormente se ha mencionado este patrón tiene relación a su vez con en el patrón *Factory Method*.

Las operaciones que define son soportadas por cada una de las clases que contiene como atributos. Su motivación a su vez consiste en reducir la complejidad del diseño y minimizar las dependencias, permitiendo que el

controlador de la aplicación no conozca detalles de la implementación y creación de los módulos.

En resumen se ha utilizado el patrón Facade para simplificar el acceso a un conjunto de clases y proporcionar en una única clase la comunicación con dicho conjunto de clases.

6.9.5 Patrón Value Object

El patrón Value Object se basa en un objeto que empaqueta datos permitiendo estructurar de forma compacta y organizada la transferencia de datos entre las distintas capas de la arquitectura de un sistema. Dicho objeto contiene todos los datos necesarios de uno o varios objetos de negocio que pueden ser accesibles mediante propiedades públicas o métodos de obtención y establecimiento de atributos (comúnmente denominados *setters* y *getters*).

Aunque un Value Object se encuentra íntimamente ligado a un objeto del dominio no se trata de los mismos objetos. Mientras que un objeto de dominio puede llegar a contener lógica de negocio, un Value Object está diseñado para contener únicamente datos, sirviendo como un mero almacén de datos.

Se ha utilizado el patrón Value Object para diferenciar los objetos que únicamente almacenan datos de configuración y que han sido automáticamente generados por JAXB, de los que aparte contienen lógica de negocio, siendo estos últimos los denominados objetos de dominio.

7.Gestión del proyecto

En las siguientes páginas se detallan la planificación y el presupuesto para el desarrollo de la plataforma de reconocimiento. Los objetivos de este capítulo son estimar tanto los recursos económicos como los humanos y establecer un plan para definir la utilización de los recursos a lo largo de las diferentes etapas de la realización del proyecto.

7.1 Plan de proyecto

El proyecto ha sido dividido en un total de seis fases o tareas. Las fases que conforman el proyecto son las siguientes:

- Planificación
- Análisis
- Diseño
- Codificación
- Documentación
- Despliegue

A continuación se describe cada una de ellas, incluyendo cual es el objetivo y las subtareas que las componen.

Planificación

En esta tarea se estudia el coste temporal y económico que conlleva la ejecución del proyecto. El objetivo es obtener un documento que incluya una estimación del tiempo y dinero asociado al proyecto.

Análisis

En esta tarea se definen, catalogan y describen minuciosamente los requisitos que contienen los detalles técnicos del proyecto. Esta tarea se debe dividir en subtareas por su gran extensión, y se proponen las siguientes:

- Identificación de las necesidades del cliente.
- Especificación de requisitos de usuario.

Diseño

En esta tarea se aborda la especificación de los componentes de la plataforma, detallando las interfaces entre ellos, la arquitectura seleccionada, y las clases que compondrán cada uno de ellos.

Esta tarea se debe dividir en subtareas por su gran extensión, y se proponen las siguientes:

- Definición de una arquitectura de software.
- Especificación de requisitos de software.

Codificación

Una vez se tienen definidos los requisitos y se ha completado la fase de diseño se procede a codificar el sistema. El objetivo de la codificación es obtener el conjunto de clases y artefactos como archivos y/o bases de datos que compondrán el sistema. Las clases deben estar documentadas para facilitar su posterior modificación.

Documentación

En esta tarea se documenta en distintos documentos la información técnica y de uso necesaria para los usuarios del software y para desarrollos futuros. La documentación debe ser generada a lo largo del proyecto y mantenerse a la par con estado de desarrollo del proyecto.

Despliegue

La fase de despliegue se corresponde con la fase final del proyecto. Dicha fase incluye la puesta en marcha de la aplicación en la plataforma seleccionada.

Seguimiento del proyecto

La fase de seguimiento del proyecto incluye las distintas reuniones mantenidas con el tutor del proyecto, teniendo como objetivo fundamental la vigilancia de todas las actividades de desarrollo del sistema construido.

7.2 Metodología

Debido a la complejidad del proyecto anteriormente mencionada, se ha utilizado el ciclo de vida en espiral. Se define el ciclo de vida de un producto software como el periodo comprendido entre que se concibe y deja de estar disponible. Se divide normalmente en fases que estructuran y organizan las etapas de concepción, desarrollo y mantenimiento del producto software.

En el modelo de ciclo de vida en espiral se evalúan las posibles alternativas de desarrollo, se escoge la que posee un riesgo más asumible y se hace un ciclo de la espiral. Si el cliente desea seguir haciendo mejoras en el producto software se vuelven a evaluar las distintas alternativas y riesgos y se realiza otra iteración sobre la espiral. Se finaliza cuando el cliente

considera válido el producto desarrollado y no sea preciso seguir mejorándolo en un nuevo ciclo.

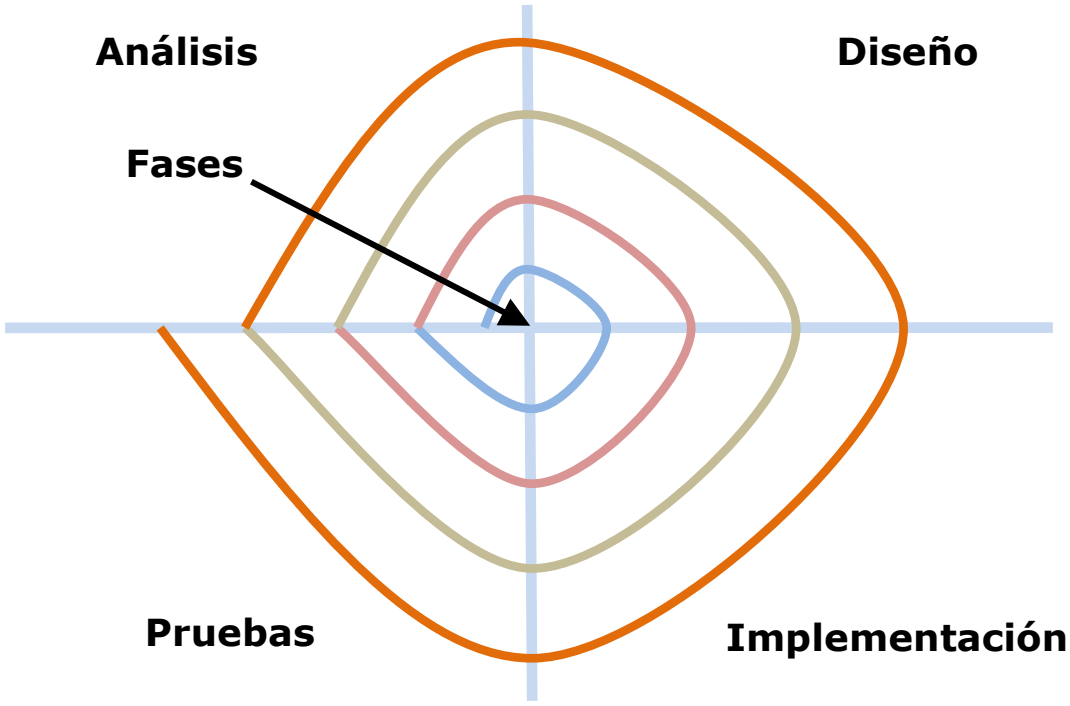


Figura 33 - Ciclo de vida en espiral

La Figura 33 muestra un diagrama que representa un desarrollo llevado a cabo en cinco fases o iteraciones del ciclo de vida. Se recomienda usar este tipo de ciclo de vida cuando se planean hacer múltiples entregas del producto software. En cada entrega se desarrollan todas las fases incorporando las experiencias recogidas en las anteriores.

Este ciclo de vida normalmente se usa en proyectos de larga duración o gran tamaño. Se utiliza también en proyectos que precisan tecnologías avanzadas o cuando es necesaria la experiencia del usuario para refinar el diseño. Las motivaciones que han llevado a la elección del ciclo de vida en espiral son las siguientes:

- **Riesgo tecnológico:** El ciclo de vida en espiral permite asumir ciertos riesgos derivados del desconocimiento tecnológico.
- **Adaptabilidad y dinamismo:** Acerca la solución del sistema de manera progresiva al permitir el desarrollo en paralelo y la especialización del equipo de desarrollo en función del componente a implementar.
- **Complejidad del sistema:** Cada uno de los componentes de la plataforma se ha abordado como un proyecto por separado, lo que ha permitido corregir anomalías y desviaciones.

7.3 Estimación de recursos temporales

A continuación se presenta una estimación de los recursos temporales divididos por cada una de las tareas definidas en la sección anterior:

Tarea	Horas
Planificación	**30**
Análisis	**100**
Diseño	**180**
Codificación	**245**
Despliegue	**1**
Documentación	**25**
Seguimiento del proyecto	**30**
TOTAL	**611**

Tabla 8 - Estimación de recursos temporales por tarea

Habiéndose establecido la fecha de comienzo el 25 de Octubre de 2010, la fecha de finalización prevista es el 24 de Septiembre de 2012.

7.4 Estimación de recursos económicos

La estimación de recursos económicos se ha dividido en varias secciones, dependiendo del tipo de las tareas que generan dicho coste. En la última sección de este apartado se resumen los costes totales incluyendo los costes derivados de cada uno de los tipos mencionados.

7.4.1 Recursos humanos

Para el cálculo de costes debidos a gastos de personal, se deben tener en cuenta los distintos perfiles que van a realizar las distintas tareas que componen el proyecto:

- Director de proyecto.
- Analista.
- Desarrollador/Programador.
- Documentador.

La fórmula que se utiliza para calcular el coste debido a recursos humanos es la siguiente:

$$Coste_{RRHH} = Ch * Dh$$

Siendo:

Ch: Coste horario.

Dh: Dedicación en horas.

A continuación se exponen brevemente los perfiles requeridos para la realización del proyecto así como sus honorarios y las aptitudes y capacidades que se esperan para ese perfil:

Director de proyecto

Requisitos: Licenciado / Ingeniero en informática

Honorarios aproximados: 105 € / hora.

Los objetivos del director de proyecto son obtener el producto a tiempo, ajustándose al presupuesto y cumpliendo con los requisitos establecidos. Por otro lado debe cumplir con las expectativas del cliente, estableciendo para ello medidas y protocolos que deberán seguir el resto del equipo

Las aptitudes y capacidades que se esperan del director de proyecto son las siguientes:

- Liderazgo para conducir al equipo a lograr los objetivos propuestos.
- Creatividad para poder tomar decisiones y acciones cuando surjan problemas.
- Experiencia y organización para distribuir los recursos de acuerdo a las tareas que se deben realizar.

Analista

Requisitos: Licenciado / Ingeniero en informática

Honorarios aproximados: 60 € / hora.

Los objetivos del analista son el determinar las necesidades esenciales y no esenciales, impedir la introducción de defectos en fases tempranas de la construcción del producto, construir el pliego de requisitos de usuario, definir la estructura básica inicial del sistema (con interacciones, interrelaciones y contextos de dicha estructura) y esbozar la especificación de la arquitectura del sistema.

Las aptitudes y capacidades personales que requiere un analista son las siguientes:

- Creatividad que le permita establecer distintos modelos de arquitectura.
- Experiencia tanto en técnicas de diseño de software como de lenguajes de programación.

- Habilidad en la comunicación, pues tendrá un trato estrecho con el cliente (alto grado de desarrollo de inteligencia emocional).

Desarrollador / Programador

Requisitos: Ingeniero técnico en informática

Honorarios aproximados: 35 € / hora.

Los objetivos del programador son disminuir la complejidad del software lo más posible. Debe intentar conseguir aumentar la eficiencia en el mantenimiento del programa, disminuir la cantidad de problemas en el testeo de los artefactos / objetos construidos, aumentar la eficiencia en el proceso de modificación del programa, reducir el tiempo de codificación y en general disminuir el costo en el ciclo de vida del software.

Las aptitudes y capacidades personales que debe poseer un programador son las siguientes:

- Experiencia en el desarrollo de aplicaciones en el ambiente seleccionado.
- Debe conocer detalladamente el lenguaje de programación en el que se va a desarrollar la aplicación.
- Debe conocer las técnicas de diseño propuestas por el analista.
- Es deseable que conozca las distintas metodologías de desarrollo existentes.

Documentador

Requisitos: No aplican.

Honorarios aproximados: 25 € / hora.

Objetivos: El objetivo principal del documentador es el mantener la información generada durante el proceso de desarrollo.

Las aptitudes y capacidades personales que debe poseer un documentador son las siguientes:

- Debe ser una persona ordenada, y ser creativo en la presentación. Además debe poseer aptitud de expresión en la escritura.
- Debe conocer tecnologías de gestión de repositorios de documentos.
- Debe conocer y utilizar el procesador de texto definido para el proyecto y ser capaz de explotar todo su potencial.

A continuación se detallan los costes debidos a recursos humanos:

Rol	Honorarios	Horas empleadas	Coste total
Director de proyecto	105€/hora	60	6.300,00€
Analista	60€/hora	280	16.800,00€
Desarrollador	35€/hora	246	8.610,00€
Documentador	25€/hora	25	625,00€
		TOTAL	**32.335,00€**

Tabla 9 - Estimación de recursos económicos por RRHH

7.4.2 Recursos materiales necesarios para el desarrollo

Para la realización del proyecto es necesaria la adquisición de ciertos materiales tangibles (hardware) e intangibles (software). La estimación incluye el periodo de amortización de estos elementos, teniendo así en cuenta el coste que suponen para este proyecto concreto.

Se estima un periodo de amortización de tres años para el hardware y de cinco años para el software.

La fórmula que se utiliza para calcular el coste de dichos materiales inventariables es la siguiente:

$$Coste_{Mat} = \left(\frac{U}{Am}\right) * Co * Pu * Uds$$

Siendo:

U: Número de meses que el equipo se va a usar en el proyecto después de la fecha de compra.

Am: Periodo de amortización del material, especificado anteriormente.

Co: Coste en euros del equipo.

Pu: Porcentaje de uso del equipo en el proyecto.

Uds: Unidades adquiridas de dicho recurso material.

A continuación se detalla la tabla de coste por material:

Concepto	Coste unitario	Uds.	Co	U	Am	Pu	$Coste_{Mat}$
Ordenador portátil Sony Vaio VGN-SR39VN	1.369,21€	3	1.369,21€	7	36	100%	798,70€
Impresora HP Laserjet 1020	199,00€	1	199,00€	7	36	100%	38,69€
Microsoft Office Home And Business	193,00€	3	193,00€	7	60	100%	67,55€
						TOTAL	**904,94€**

Tabla 10 - Estimación de recursos económicos por material

***Nota:** En la tabla no se incluye el coste del sistema operativo (Windows 7) pues viene instalado y licenciado para su uso en el equipo.

7.4.3 Materiales fungibles

Dada la dificultad para calcular con exactitud el coste del material fungible que se utilizará en el proyecto, se estimará como el 0,5% del coste atribuido a recursos humanos.

Concepto	Porcentaje sobre Coste de RRHH	Coste de Personal	Total
Material Fungible	0,5%	32.335,00€	161,67€

Tabla 11 - Estimación de recursos económicos en materiales fungibles

7.4.4 Gastos indirectos

Se consideran gastos indirectos los costes derivados de los recursos humanos y materiales necesarios para la explotación. Algunos de estos gastos son luz, agua, seguridad, conexión a internet, servicios de limpieza y otros gastos generales.

Como gastos indirectos se atribuye a este proyecto un 15% de los costes atribuidos a recursos humanos.

Concepto	Porcentaje sobre Coste de RRHH	Coste de Personal	Total
Gastos indirectos	15%	32.335,00€	4850,25€

Tabla 12 - Estimación de recursos económicos en gastos indirectos

7.4.5 Resumen de costes

En este apartado se resumen los costes asociados a los distintos conceptos del proyecto. A continuación se detalla la tabla de costes totales y se muestra la cantidad final:

Concepto	Coste
Recursos Humanos	32.335,00€
Materiales necesarios para el desarrollo	904,94€
Materiales fungibles	161,67€
Gastos indirectos	4850,25€
TOTAL DESARROLLO	38.251,86€
Cobertura de riesgos (15%)	5.737,77€
TOTAL ANTES DE BENEFICIOS	43.989,63€
Beneficios empresariales (20%)	8.797,92€
TOTAL ANTES DE IMPUESTOS	52.787,55€
Impuestos (21%)	11085,38€
TOTAL	**63.872,93€**

Tabla 13 - Estimación de costes totales

Al proyecto se le estima un presupuesto de *Sesenta y tres mil ochocientos setenta y dos euros y noventa y tres céntimos*.

8. Conclusiones

A continuación se exponen las propuestas de trabajo futuro y las conclusiones a las que se ha llegado con la realización de este proyecto. También se incluye una sección dedicada a las conclusiones personales.

8.1 Propuestas de trabajo futuro

En esta sección se enumeran algunas propuestas para futuros trabajos que inicialmente no fueron incluidas en el alcance de este proyecto y que durante el desarrollo del mismo se estimaron importantes:

- Dar soporte al aprendizaje de los módulos, que permita a partir del feedback aportado por el usuario de la aplicación la mejora de los resultados de los módulos implementados.
- Extender la interfaz común de los *invokers* permitiendo mediante frameworks de programación basados en JNI [Liang, 1999], Jython [Pedroni, et al. 2002] ó JPype [Menard, 2012] para permitir módulos escritos en otros lenguajes como por ejemplo C, C++ o Python. El objetivo de dicha propuesta sería ampliar el abanico de módulos soportados, extendiendo el desarrollo de los módulos a otros lenguajes y plataformas.
- Ampliación de la funcionalidad del módulo proveedor para que admita también audio en tiempo real, cuya fuente sería la interfaz de entrada de la tarjeta de audio (donde se podría conectar un micrófono por ejemplo).
- Agregar nuevos soportes para los archivos de configuración, permitiendo por ejemplo guardarlos en bases de datos.
- Mejorar el módulo proveedor, permitiendo la transformación de muestras de audio contenidas en archivos comprimidos (como MP3 ó Ogg) a PCM. Sería necesario añadir una etapa adicional en el módulo proveedor.
- Crear un terminal de consola y los comandos de manejo de la misma que permitan explotar la potencialidad completa del sistema. Esto asemejaría el modo de funcionamiento al utilizado en HTK, descrito en la sección 3.3. Para ello sería necesario sustituir la capa Vista del patrón arquitectónico MVC.
- Añadir más controles a la interfaz no relacionados directamente con la transcripción como pudiera ser un control de volumen o de ecualización de la señal de entrada.
- Permitir salvar configuraciones manuales para su posterior carga como configuración preprogramada.

- Permitir deshacer y rehacer cambios sobre la configuración, almacenando un histórico de configuraciones en lugar de sólo la última. Esto podría conseguirse sustituyendo el patrón de diseño memento por un patrón Command.

8.2 Conclusiones

Las conclusiones que se presentan a continuación son el resultado de comparar los objetivos planteados al comienzo de este proyecto y los objetivos conseguidos a su finalización.

Una de las tareas más complicadas en este proyecto ha sido la definición del alcance, pues inicialmente este proyecto iba a servir de base para la realización de otros tres proyectos finales de carrera. También ha resultado complicada la comprensión del dominio, pues este no compone la definición de ninguna de las asignaturas de la carrera de ingeniería informática, y es un campo que los autores e investigadores citados en este proyecto consideran complejo.

Los objetivos que se han cubierto con la realización del presente proyecto son los siguientes:

- Se ha conseguido desarrollar una plataforma que permite utilizar en tiempo de ejecución módulos diseñados de forma independiente.

- Se ha conseguido tratar la señal de entrada de audio generando un flujo de bytes que es proporcionado al módulo encargado de la extracción de características.

- Se ha logrado obtener un conjunto de estadísticos que describen la calidad de un sistema de reconocimiento del habla constituido por la integración de los módulos anteriormente citados.

- Se ha conseguido leer los parámetros de configuración desde un archivo en XML y también la posterior edición de estos valores desde la plataforma, con la dificultad que plantea el intercambio de datos con módulos de los cuales se desconoce su implementación.

- Se han utilizado distintas tecnologías y componentes que han facilitado la realización del proyecto en gran parte evitando el tener que desarrollar funciones comúnmente utilizadas.

Se han podido aplicar conocimientos adquiridos durante la carrera en cuanto a gestión de proyectos, análisis y diseño de software. También se han aplicado conocimientos adquiridos durante mi vida laboral. El grado de cumplimiento de los objetivos inicialmente planteados puede considerarse como satisfactorio. Este proyecto va a permitir la realización de experimentos con módulos de reconocimiento independientes, y evaluar su resultado, pudiendo servir esta plataforma como base de futuros proyectos finales de carrera.

8.3 Conclusiones personales

Debo reconocer que al comienzo del proyecto subestimé su dificultad y el tiempo que me iba a requerir. He empleado todo el tiempo que he podido (el que me ha dejado el trabajo) en su realización durante estos casi dos años, lo que ha supuesto renunciar en muchas ocasiones a vacaciones y fines de semana. Pero a cambio he aprendido a asumir todos los roles durante la realización de un proyecto de ingeniería del software, conocer sobre el dominio del reconocimiento del habla (para mi desconocido hasta ahora) y sobre todo a aprender nuevas tecnologías que sin duda me ayudarán en el desempeño de mi actividad profesional.

En el proyecto personalmente he pasado por distintas fases. Al comienzo tomé el proyecto con mucha ilusión. En muchos momentos ha llegado la desesperación por no ver el final y por la exigencia del tutor, que espero se vea reflejada en la calidad del proyecto. Y una vez redactada la memoria y sin haberla presentado aún llega la incertidumbre por la nuevos retos que me esperan una vez finalizada la carrera.

En definitiva estoy contento con el trabajo que he realizado, y creo que el esfuerzo ha merecido la pena.

9.Referencias Bibliográficas

[Anduaga, et al. 2006] Anduaga Márquez, Gilberto, y Héctor Caudel García. «Comentarios sobre el software de código abierto Sphinx.» Conciencia Tecnológica. nº 032. Aguascalientes, 2006.

[ASF, 2012] Apache Software Foundation. «Apache Commons.» sitio web oficial de Apache Commons. http://commons.apache.org/ (último acceso: 09 de 01 de 2012).

[B. Shelly, et al. 2012] B. Shelly, Gary, y Harry J. Rosenblatt. Systems Analysis and Design. Novena. Cengage Learning, 2012.

[Balsamiq, 2008] Balsamiq Studios. sitio web de Balsamiq. 06 de 2008. http://www.balsamiq.com/ (último acceso: 22 de 06 de 2011).

[Barranco, 2001] Barranco de Areba, Jesús. Metodología del análisis estructurado de sistemas. Madrid: Universidad Pontificia de Comillas, 2001.

[Cambridge, 2003] Cambridge University Engineering Department. «HTK Speech Recognition Toolkit.» What is HTK? http://htk.eng.cam.ac.uk/ (último acceso: 12 de 05 de 2012).

[CMU, 2002] Carnegie Mellon University. «Definición de la Clase NISTAlign en la API proporcionada por sphinx4.» sitio web oficial de Sphinx en SourceForge. http://cmusphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/util/NISTAlign.html (último acceso: 2012 de 07 de 02).

[CMU, 2008] «Sphinx-4: A speech recognizer written entirely in the JavaTM programming language.» página de descarga de Sphinx-4 en la plataforma SourceForge. 2008. http://cmusphinx.sourceforge.net/sphinx4/ (último acceso: 1 de Junio de 2012).

[Carillo, 2007] Carrillo Aguilar, Roberto. «Diseño y manipulación de modelos ocultos de Markov, utilizando herramientas HTK. Una tutoría.» Ingeniare. Revista Chilena de Ingeniería. 15, nº 1 (2007): 18-26.

[Carpenter, et al. 1991] Carpenter, Gail A. y Grossberg, Stephen. 1991. Pattern recognition by self-organizing neural networks. Massachusetts: Massachusetts Institute of Technology, 1991. 9780262031769.

[Casacuberta, 1987] Casacuberta, Francisco, y Enrique Vidal. Reconocimiento automático del habla. Barcelona: Marcombo, 1987.

[Chunrong, et al.] Chunrong Lai, Shih-Lien Lu, y Zhao Qingwei. «Performance Analysis of Speech Recognition Software.»

[Colebourne, 2012] Colebourne, Stephen, y Brian O'Neill. http://joda-time.sourceforge.net/ (último acceso: 28 de 02 de 2012).

[Cook, 2002] Cook, Stephen. 2002. The Linux Documentation Project. Speech Recognition HOWTO. [Online] 2002. [Último acceso: 6 de Agosto 2012.] http://www.tldp.org/HOWTO/Speech-Recognition-HOWTO/index.html.

[E. Krasner, et al. 1988] E. Krasner, Glenn, y Stephen T. Pope. «A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System.» California, 1988.

[Eckel, 2002] Eckel, Bruce. Piensa en Java. Segunda. Prentice Hall, 2002.

[Eclipse, 2012] Eclipse Foundation. http://www.eclipse.org/ (último acceso: 07 de 07 de 2011).

[Engel, 1999] Engel, Joshua. Programming for the Java Virtual Machine. Addison-Wesley, 1999.

[Fandiño, 2005] Fandiño Rodríguez, Deiby Alexander. «Estado del arte en el reconocimiento Automático de voz.» 2005.

[Ferrández, et al. 2004] Ferrández, Antonio, y otros. Tecnologías del texto y del habla. Editado por M. Antonia Martí y Joaquim Llisterri. Barcelona: Edicions Universitat de Barcelona, 2004.

[Gamma, et al. 2006] Gamma, Erich, Richard Helm, Ralph Jhonson, y John Vlissides. Patrones de diseño. Addison Wesley, 2006.

[Gosling, et al. 2000] Gosling, James, Bill Joy, Guy Steele, y Bracha Gilad. The Java Language Specification. California: Addison-Wesley, 2000.

[Hierro, 2004] Hierro Álvarez, Jorge. «Informe técnico sobre los sistemas de reconocimiento de voz.» Departamento de prensa, Assit, Madrid, 2004.

[Hopfe, et al. 2010] J. Hopfe, Christina, Yacine Rezgui, Elisabeth Métais, Alun Preece, y Haijiang Li. Natural Language Processing and Information Systems. Nueva York: Springer, 2010.

[Hualde, et al. 2002] Hualde, José Ignacio, Olarrea, Antxón and Escobar, Anna María. 2002. Introducción a la lingüística hispánica. Cambridge: Cambridge University Press, 2002. 9780521803144.

[Huang, et al. 2001] Huang, Xuedong, Alex Acero, and Hsiao-Wuen Hon. Spoken Language Processing - A guide to Theory, Algorithm and System Development. New Jersey: Prentice Hall, 2001.

JavaZoom. «MP3 SPI Support.» Sitio web oficial de MP3 SPI. http://www.javazoom.net/mp3spi/documents.html (último acceso: 02 de 02 de 2012).

—. «Vorbis SPI Project.» sitio web oficial sobre Vorbis SPI. http://www.javazoom.net/vorbisspi/vorbisspi.html (último acceso: 02 de 03 de 2012).

[L. Winblad, et al. 1993] L. Winblad, Ann, Samuel D. Edwards, y David R. King. Software orientado a objetos. Traducido por Ramón Ruiz Ayuso y Luis Joyanes Aguilar. Addison-Wesley / Diaz de Santos, 1993.

[Lamere et al. 2004] Lamere, Paul, y otros. «The CMU Sphinx-4 Speech Recognition System.» 2004.

[Liang, 1999] Liang, Sheng. The Java Native Interface - Programmer's Guide and Specification. Addison-Wesley, 1999.

[Loy, et al. 2003] Loy, Marc, Robert Eckestein, Dave Wood, James Elliot, y Brian Cole. Java Swing. Segunda. O'Reilly, 2003.

[Martínez, 1998] Martínez Celdrán, Eugenio. 1998. Lingüistica: Teoría y aplicaciones. Barcelona: Masson, 1998. 9788445807255.

[Menard, 2012] Menard, Steve. «JPype - Bridging the worlds of Java and Python.» sitio web de JPype en SourceForge. http://jpype.sourceforge.net/ (último acceso: 12 de 06 de 2012).

[Microsoft, 2012] Microsoft Corporation. «Sitio de descarga de Microsoft Speech SDK.» Microsoft. http://www.microsoft.com/en-us/download/details.aspx?id=10121 (último acceso: 02 de 06 de 2012).

[Morales, et al. 2007] Morales España, Germán Andrés, René Alexander Barrera Cárdenas, y Juan José Mora Flórez. «Reconocimiento de comandos por voz con máquinas de soporte vectorial a través de bandas espectrales.» Scientia et Technica, Diciembre 2007: 1.

[Nuance, 2012] Nuance. «Dragon Speech Recognition Software Homepage.» http://www.nuance.com/dragon/index.htm (último acceso: 21 de 05 de 2012).

[Oracle, 2012] http://netbeans.org/ (último acceso: 13 de 07 de 2011).

—. http://jaxb.java.net/ (último acceso: 26 de 01 de 2012).

—. «Java Sound Programmer Guide.» sitio web oficial de Java Sound. http://docs.oracle.com/javase/1.5.0/docs/guide/sound/programmer_guide/contents.html (último acceso: 01 de 02 de 2012).

[Pedroni, et al. 2002] Pedroni, Samuele, y Noel Rappin. Jython Essentials. O'Reilly, 2002.

[Revuelta, et al. 2008] Subtitulado Cerrado para la Accesibilidad de Personas con Discapacidad. Revuelta Sanz, Pablo, et al. 2008. 41, Madrid: s.n., 2008, Procesamiento del lenguaje Natural, pp. 305-306.

[Rodríguez, et al. 2004] Rodríguez, Santiago y Smith-Agreda, José María. 2004. Anatomía de los órganos del lenguaje, visión y audición. Madrid: Panamericana, 2004. 9788479038687.

[Saalfeld, 1999] Saafeld, Alan. «Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm.» Cartography and Geographic Information Science 26, nº 1 (1999): 7-18.

[Salcedo, 2007] Salcedo Campos, Francisco Javier. Modelos Ocultos de Markov del reconocimiento de voz a la música. Lulu, 2007.

[Sánchez, 1995] Sánchez Blanco, María Dulce. 1995. La lengua y su didáctica. Murcia: Universidad de Murcia, 1995. 9788476844885.

[Singh, 2003] Singh, Rita. «Encyclopaedia of Human Computer Interaction.» sitio web de la univerdad Carnegie Mellon. 10 de Octubre de 2003. http://www.cs.cmu.edu/~rsingh/homepage/sphinx_history.html (último acceso: 25 de Mayo de 2012).

[Suárez, 2007] Suárez, C., et al. 2007. Tratado de Otorrinolaringología y Cirugía de Cabeza y Cuello. Madrid : Editorial Médica Panamericana, 2007. 978-84-9835-076-0

[Tamba, 2005] Tamba Mecz, Irene. 2005. La semántica. México : Fondo de cultura económica de España, S.L, 2005. 9789681673109.

[Viterbi, 1967] Viterbi, Andrew James. «Error Bounds for Convolutional Codes and an Asymptotically Optimum Algorithm.» IEEE Trans. Information Theory, Abril 1967.

[W3C, 2003] W3C. «Sitio web de W3C sobre XML.» W3C. http://www.w3.org/XML/ (último acceso: 06 de 04 de 2012).

[Young, 2007] Young, Steve. ATK - An application Toolkit for HTK. Cambridge, 2007.

[Young, et al. 2009] Young, Steve, y otros. The HTK Book (for HTK Version 3.4). 2009.

[Zhelezniakov, et al.] Zhelezniakov, Peter, y otros. «Community Development of Java Technology Specifications - JSR 295: Beans Binding.» sitio web de Java Community Process. Oracle. http://jcp.org/en/jsr/detail?id=295 (último acceso: 05 de 06 de 2012).

Apéndice I: Glosario de términos

Vocabulario

Alófonos: Los alófonos son sonidos distintos o realizaciones diferentes de un mismo fonema. Se encuentran por tanto dentro del campo de la Fonética.

Aplicación de escritorio: Una aplicación de escritorio es aquella que es capaz de ejecutarse de manera autónoma en un ordenador de sobremesa o portátil, a diferencia de las denominadas "Aplicaciones Web" que requieren un navegador web para funcionar.

Classpath: El Classpath es una variable de entorno que indica al JDK dónde debe buscar los archivos a compilar o ejecutar, permitiendo no tener que escribir la ruta completa.

Distancia de Levenshtein: También llamado distancia de edición o entre palabras, indica el número mínimo de ediciones que se necesitan para transformar una cadena de caracteres en otra. Las operaciones que se consideran son inserción, eliminación o sustitución de un carácter.

Framework: Un Framework es un conjunto de clases cooperantes que constituyen un diseño reutilizable para una clase específica de software. Un Framework determina la arquitectura de la aplicación, la estructura general, su partición en clases y objetos, y sus colaboraciones.

Lemario: Conjunto de los lemas o entradas que contiene un repertorio lexicográfico.

Métodos estáticos: Los métodos estáticos (también llamados métodos de clase) se cargan en memoria en tiempo de compilación en lugar de cargarse a medida que se ejecutan las líneas de código del programa. Para invocar a un método estático no se necesita crear un objeto de la clase que lo contiene. La API de Java contiene un gran número de métodos estáticos, por ejemplo en la clase Math del paquete java.lang.

Mockup: Un mockup (también se puede encontrar escrito como mock-up) es un modelo a escala o tamaño real usado en el diseño de un programa o dispositivo para evaluación del diseño o como demostración. Es un prototipo que proporciona una pequeña parte de la funcionalidad del sistema que se desea construir. El caso más común en desarrollo del software es usarlo para definir la interfaz de usuario, para mostrar al usuario final como lucirá la aplicación.

Módulo de reconocimiento: Parte autónoma, repetitiva e intercambiable que se carga en la plataforma y que realiza una tarea en el proceso de reconocimiento automático del habla. Un módulo recibe como entrada la salida de otro, salvo los módulos que son entrada o salida del sistema, que sólo tendrán una salida o entrada conectada con otro módulo respectivamente.

Módulo antecesor: Módulo de reconocimiento situado inmediatamente antes de uno dado y al que invoca operaciones sobre las que espera un resultado.

Módulo sucesor: Módulo de reconocimiento situado inmediatamente después de uno dado cuyo cometido es realizar las operaciones que le son invocadas devolviendo un resultado.

Namespace: Recurso utilizado para proveer de nombres únicos a los elementos contenidos en un documento XML definido en las recomendaciones de la W3C. El namespace name definido para un namespace es una URL que donde se ubica el recurso que define el vocabulario. Suele encontrarse bajo el control del autor o de la organización y por ello se localiza en el servidor Web del autor de la organización.

Signatura de un método: También llamada firma de un método, define el tipo de entrada y de salida del método. Incluye al menos el nombre del método (o función) y el número de sus parámetros. En algunos lenguajes de programación, puede incluir el tipo de datos que devuelve la función aparte del tipo de sus parámetros.

Toolkit: Conjunto de clases relacionadas y reutilizables diseñadas para proporcionar funcionalidad útil de propósito general. Los Toolkits no imponen un diseño particular en una aplicación sino que ayudan a que la aplicación cumpla con su cometido.

Widget: Un Widget es una pequeña aplicación cuyo objetivo es dar fácil acceso a funciones usadas frecuentemente.

Acrónimos

AAC: Advanced Audio Coding.

API: Application Programming Interface.

ARPA: Advanced Research Projects Agency.

ASR: Automatic Speech Recognition.

ATK: Application Toolkit for HTK.

AWT: Abstract Window Toolkit.

COM: Component Object Model.

DFD: Diagrama de flujo de datos

DOM: Document Object Model.

DTW: Dynamic Time Warping.

FLAC: Free Lossless Audio Codec.

GMM: Gaussian Mixture Model.

GNU: Acrónimo recursivo de GNU Not Unix.

GPL: General Public License.

GPS: Global Positioning System.

GUI: Graphical User Interface.

HMM: Hidden Markov Models.

HTK: Hidden Markov Model Toolkit.

IDE: Integrated Development Environment.

J2SE: Java 2 Platform, Standard Edition.

JAXB: Java Architecture for XML Binding.

JCP: Java Community Process.

JNI: Java Native Interface.

JSR: Java Specification Request.

JVM: Java Virtual Machine.

LM: Language Model.

MIDI: Musical Instrument Digital Interface.

MP3: MPEG-1 Audio Layer III ó MPEG-2 Audio Layer III.

MPEG: Moving Picture Experts Group.

MVC: Modelo Vista Controlador.

OOXML: Office Open XML.

PCM: Pulse-Code Modulation.

PDA: Personal Digital Assistant.

RRHH: Recursos Humanos.

RS: Requisito de Software.

RU: Requisito de Usuario.

SAPI: Speech Application Programming Interface.

SAX: Simple API for XML.

SDK: Software Development Kit.

SOAP: Simple Object Access Protocol.

SPI: Service Provider Interface.

SPL: Sound Pressure Level.

UML: Unified Modeling Language.

URL: Uniform Resource Locator.

W3C: World Wide Web Consortium.

WAV: Apócope de WAVEForm audio file format.

WER: Word Error Rate.

WSDL: Web Services Description Language.

XBRL: Extensible Business Reporting Language.

XML: Extensible Markup Language.

XPath: XML Path Language.

XQuery: XML Query Language.

XSD: XML Schema Definition.

XSLT: Extensible Stylesheet Language Transformations.

Apéndice II: Especificación de requisitos

Requisitos de usuario

En esta sección se amplían los requisitos definidos de manera no formal en la sección "Capacidades y restricciones generales".

Se consideran los siguientes atributos:

- **Nombre:** Nombre del requisito.
- **Prioridad:** Referencia al orden temporal del requisito frente a los demás.
- **Necesidad:** Esta variable indica si es un requisito indispensable para el usuario o no.
- **Estabilidad:** Posibilidad que el requisito cambie a lo largo del tiempo.
- **Antecesor:** Requisito del que deriva el requisito mencionado.
- **Descripción:** Descripción del requisito.

A continuación se describen detalladamente los requisitos de usuario:

Identificador: RU-01
Nombre: Objetivo del sistema
Prioridad: Alta
Necesidad: Alta
Estabilidad: Alta
Descripción: El objetivo del sistema es la transcripción automática del audio proporcionado en texto, basándose dicha transcripción en los resultados proporcionados por los módulos seleccionados por el usuario.

Identificador: RU-02
Nombre: Distintos tipos de módulos
Prioridad: Alta
Necesidad: Alta
Estabilidad: Alta
Antecesor: RU-01
Descripción: El usuario podrá cargar en la plataforma distintos módulos que cumplan con una interfaz definida para la extracción de características de una muestra, el reconocimiento del locutor, la identificación de fonemas, la identificación de palabras y la identificación de frases.

Identificador: RU-03
Nombre: Selección de un módulo para cada etapa
Prioridad: Media
Necesidad: Media
Estabilidad: Alta
Antecesor: RU-02
Descripción: Cada etapa de reconocimiento del habla será procesada únicamente por un módulo.

Identificador: RU-04
Nombre: Ejecución en tiempo real
Prioridad: Media
Necesidad: Media
Estabilidad: Alta
Antecesor: RU-01
Descripción: El sistema proporcionará la transcripción del fichero elegido en tiempo real.

Identificador: RU-05
Nombre: Configuración de los módulos
Prioridad: Media
Necesidad: Alta
Estabilidad: Alta
Antecesor: RU-02
Descripción: El usuario podrá seleccionar valores de configuración para los módulos seleccionados.

Identificador: RU-06
Nombre: Almacenamiento de los diferentes valores de configuración
Prioridad: Media
Necesidad: Media
Estabilidad: Alta
Antecesor: RU-05
Descripción: Los valores de configuración deben poder almacenarse de modo que luego se puedan seleccionar cualquiera de estas configuraciones preprogramadas.

Identificador: RU-07
Nombre: Elección de los valores de configuración
Prioridad: Baja
Necesidad: Media
Estabilidad: Alta
Antecesor: RU-05
Descripción: Los módulos proporcionarán una configuración por defecto, que será la utilizada en caso de que el usuario no la proporcione.

Identificador: RU-08
Nombre: Limitar ambigüedad en la elección de los módulos
Prioridad: Media
Necesidad: Alta
Estabilidad: Media
Antecesor: RU-02
Descripción: El sistema impedirá que el usuario seleccione un módulo del tipo incorrecto, e informará al usuario de dicho error.

Identificador: RU-09
Nombre: Módulos de prueba
Prioridad: Baja
Necesidad: Media
Estabilidad: Alta
Antecesor: RU-02
Descripción: Junto con la plataforma deben desarrollarse cinco módulos de prueba (que simularán el funcionamiento de un módulo real) para poder verificar el correcto funcionamiento del sistema.

Identificador: RU-10
Nombre: Información ofrecida al usuario
Prioridad: Media
Necesidad: Alta
Estabilidad: Alta
Antecesor: RU-01
Descripción: El sistema ofrecerá información sobre características de la señal de audio, los sucesos y eventos reseñables que ocurran en el sistema (comienzo y finalización de acciones, errores) y sobre la ejecución de la transcripción.

Identificador: RU-11
Nombre: Información sobre el fichero que contiene la muestra de audio
Prioridad: Media
Necesidad: Baja
Estabilidad: Media
Antecesor: RU-10
Descripción: El sistema mostrará la siguiente información si la entrada proporcionada es un archivo de audio: • Tipo de señal de audio • Frecuencia de muestreo • Número de bits que forman cada muestra • Si la muestra de audio es mono o estéreo • Formato interno de representación (Little Endian o Big Endian)

Identificador: RU-12
Nombre: Información acerca de la transcripción
Prioridad: Alta
Necesidad: Alta
Estabilidad: Media
Antecesor: RU-10
Descripción: El sistema mostrará dos tipos de información relacionadas con la transcripción: • Datos estadísticos. • De progreso.

Identificador: RU-13
Nombre: Información acerca de la transcripción
Prioridad: Alta
Necesidad: Alta
Estabilidad: Media
Antecesor: RU-10
Descripción: Al comparar la transcripción resultado con la transcripción objetivo, el sistema proporcionará los siguientes datos estadísticos • Número de palabras añadidas. • Número de palabras eliminadas. • Número de palabras modificadas. • Número de palabras acertadas. • Exactitud de la palabra (valor entre 0 y 1). • Tasa de error de la palabra (WER, siglas en inglés de Word Error Rate). • Distancia de Levenshtein.

Identificador: RU-14
Nombre: Información acerca del progreso de la transcripción
Prioridad: Media
Necesidad: Alta
Estabilidad: Alta
Antecesor: RU-13
Descripción: El sistema informará de la duración total y del progreso de la transcripción en relación al total, mostrando el progreso en formato hh:mm:ss (donde "hh" son horas, "mm" son minutos y "ss" son segundos)

Identificador: RU-15
Nombre: Tipo de modulación de la señal de audio
Prioridad: Alta
Necesidad: Media
Estabilidad: Media
Antecesor: RU-01
Descripción: El tipo de modulación de la señal de audio con el que tratará el sistema será PCM.

Requisitos de software

En esta sección se amplían las propiedades descritas en la sección 3.5 "Catálogo de requisitos de software".

Se consideran los siguientes atributos:

- **Nombre:** Nombre del requisito
- **Necesidad:** Esta variable indica si es un requisito indispensable para el usuario o no.
- **Prioridad:** Referencia al orden temporal del requisito frente a los demás.
- **Fuente:** Indica si el requisito ha sido propuesto por el tutor del proyecto o por el alumno.
- **Trazabilidad:** Requisito de usuario que motiva la creación del requisito de software.
- **Descripción:** Descripción del requisito.

A continuación se describen detalladamente los requisitos de software:

IDENTIFICADOR: RS-01
Nombre: Resultados de la transcripción
Necesidad: Esencial
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-01, RU-04
Descripción: Debe existir un área de texto en la pantalla principal para mostrar los resultados de la transcripción. Dicho área de texto se actualizará cada vez que el módulo sintáctico produzca resultados.

IDENTIFICADOR: RS-02
Nombre: Errores en la plataforma
Necesidad: Deseable
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-08, RU-10
Descripción: Debe existir un área de texto en la pantalla principal a modo de consola para mostrar los errores que ocurran en el sistema.

IDENTIFICADOR: RS-03
Nombre: Barra de progreso de duración
Necesidad: Deseable
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-10, RU-12
Descripción: Debe existir una barra de progreso para mostrar la duración de la parte transcrita en la pantalla principal. Inicialmente la barra de progreso estará vacía, y se representará llena cuando finalice la transcripción.

IDENTIFICADOR: RS-04
Nombre: Periodo de actualización de la barra de progreso
Necesidad: Deseable
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-10, RU-12
Descripción: El periodo de actualización de la barra de duración será de 1 segundo.

IDENTIFICADOR: RS-05
Nombre: Etiqueta de duración
Necesidad: Deseable
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-10, RU-12, RU-14
Descripción: La pantalla principal mostrará una etiqueta donde se presentará la duración del archivo y la duración de la parte transcrita en formato hh:mm:ss (donde "hh" son horas, "mm" son minutos y "ss" son segundos).

IDENTIFICADOR: RS-06
Nombre: Controles de la transcripción
Necesidad: Esencial
Prioridad: Alta
Fuente: Alumno
Trazabilidad: RU-01, RU-04
Descripción: Los controles de la transcripción que deben mostrarse en la pantalla principal son los siguientes: botón de inicio, botón de pausa y botón de detención de la transcripción.

IDENTIFICADOR: RS-07
Nombre: Lenguaje de programación
Necesidad: Esencial
Prioridad: Alta
Fuente: Alumno
Trazabilidad: RU-01
Descripción: La aplicación se programará con el lenguaje de programación Java.

IDENTIFICADOR: RS-08
Nombre: Ejecución multiplataforma
Necesidad: Opcional
Prioridad: Media
Fuente: Alumno
Trazabilidad: -
Descripción: La plataforma podrá ejecutarse en sistemas que posean una máquina virtual de java instalada.

IDENTIFICADOR: RS-09
Nombre: Biblioteca gráfica
Necesidad: Esencial
Prioridad: Alta
Fuente: Alumno
Trazabilidad: RU-01
Descripción: El desarrollo de la interfaz gráfica de usuario se realizará usando la biblioteca gráfica Swing.

IDENTIFICADOR: RS-10
Nombre: Conectividad
Necesidad: Deseable
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-02, RU-03, RU-08
Descripción: Es necesaria conexión de área local si se desea cargar un módulo que esté almacenado en un ordenador presente en la red de área local.

IDENTIFICADOR: RS-11
Nombre: Ejecución en red
Necesidad: Deseable
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-02, RU-03, RU-08
Descripción: Si se desea escoger un módulo almacenado en un servidor, este debe estar almacenado en una ruta accesible y con permiso de lectura.

IDENTIFICADOR: RS-12
Nombre: Ventana de selección de módulos
Necesidad: Esencial
Prioridad: Alta
Fuente: Alumno
Trazabilidad: RU-02, RU-03, RU-08
Descripción: La ventana de selección de módulos mostrará únicamente los que contengan la extensión *".jar"*. Se mostrarán las carpetas a efectos de navegación por el sistema de archivos, pero no de selección.

IDENTIFICADOR: RS-13
Nombre: Validación de módulos
Necesidad: Esencial
Prioridad: Alta
Fuente: Alumno
Trazabilidad: RU-03, RU-08
Descripción: Se debe mostrar un mensaje de error por la consola si existen errores relacionados con el formato o tipo de módulo seleccionado por el usuario.

IDENTIFICADOR: RS-14
Nombre: Classpath de la plataforma
Necesidad: Esencial
Prioridad: Alta
Fuente: Alumno
Trazabilidad: RU-01, RU-02, RU-03, RU-08
Descripción: Debe ser posible la incorporación de clases en el Classpath en tiempo de ejecución.

IDENTIFICADOR: RS-15
Nombre: Interfaz para los módulos de usuario
Necesidad: Esencial
Prioridad: Alta
Fuente: Alumno
Trazabilidad: RU-02, RU-03, RU-08
Descripción: La interfaz java debe identificar la signatura de las operaciones que tienen que implementar los módulos desarrollados por el usuario.

IDENTIFICADOR: RS-16
Nombre: Módulos de prueba
Necesidad: Deseable
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-09
Descripción: Los módulos de prueba deben implementar la interfaz de los módulos y devolver resultados ficticios.

IDENTIFICADOR: RS-17
Nombre: Ventana de selección de archivos de audio
Necesidad: Esencial
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-15
Descripción: La ventana de selección de archivos de audio deberá mostrará únicamente los que contengan la extensión "*.wav*". Se mostrarán las carpetas a efectos de navegación por el sistema de archivos, pero no de selección.

IDENTIFICADOR: RS-18
Nombre: Procesador de audio
Necesidad: Esencial
Prioridad: Alta
Fuente: Alumno
Trazabilidad: RU-15
Descripción: El procesador de audio deberá convertir la información contenida en un fichero de audio WAV en formato PCM en una colección de bytes.

IDENTIFICADOR: RS-19
Nombre: Validación del fichero de audio
Necesidad: Deseable
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-15
Descripción: Se debe mostrar un mensaje de error por la consola si existe un error con el formato o el tipo de fichero de audio seleccionado.

IDENTIFICADOR: RS-20
Nombre: Área de propiedades del archivo de audio
Necesidad: Opcional
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-10, RU-11
Descripción: Debe existir un área de texto en la pantalla principal donde mostrar propiedades del fichero de audio. La información mostrada será la siguiente: • Tipo de señal de audio. • Frecuencia de muestreo. • Número de bits que forman cada muestra. • Si la muestra de audio es mono o estéreo. • Formato interno de representación (Little Endian o Big Endian).

IDENTIFICADOR: RS-21
Nombre: Gráfica de amplitud
Necesidad: Opcional
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-10, RU-12
Descripción: Debe mostrarse en la pantalla principal una gráfica que represente la amplitud de la señal de audio respecto del tiempo.

IDENTIFICADOR: RS-22
Nombre: Ventana de configuración de módulos
Necesidad: Esencial
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-05, RU-06, RU-07
Descripción: La ventana de configuración de módulos debe mostrar tanto una ventana de selección de archivos para la selección de configuraciones preprogramadas como un formulario de introducción de datos para la configuración manual de los módulos.

IDENTIFICADOR: RS-23
Nombre: Validación de datos del formulario
Necesidad: Deseable
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-05, RU-06
Descripción: Se deben validar los datos de entrada introducidos por el usuario cuando el modo de configuración sea manual, mostrando un mensaje de error en la parte superior del formulario en caso de que los datos sean incorrectos.

IDENTIFICADOR: RS-24
Nombre: Configuración de módulos
Necesidad: Esencial
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-05, RU-06, RU-07
Descripción: Es preciso que los módulos se hayan seleccionado antes de que se activen los controles de configuración de módulos.

IDENTIFICADOR: RS-25
Nombre: Configuración preprogramada
Necesidad: Deseable
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-06
Descripción: Las configuraciones preprogramadas se almacenarán en archivos XML.

IDENTIFICADOR: RS-26
Nombre: Configuración por defecto
Necesidad: Esencial
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-07
Descripción: Los módulos podrán incluir una ruta de una configuración preprogramada por defecto, que será la seleccionada por defecto si el usuario no proporciona una configuración distinta.

IDENTIFICADOR: RS-27
Nombre: Ventana de obtención de estadísticos de la transcripción
Necesidad: Esencial
Prioridad: Baja
Fuente: Alumno
Trazabilidad: RU-12, RU-13
Descripción: Existirá una ventana emergente que mostrará los estadísticos de la transcripción una vez finalizada esta. Si no se han realizado la transcripción los valores de los estadísticos estarán a cero. Los estadísticos que se mostrarán serán los siguientes: • Número de palabras añadidas. • Número de palabras eliminadas. • Número de palabras modificadas. • Número de palabras acertadas. • Exactitud de la palabra (valor entre 0 y 1). • Tasa de error de la palabra (WER, siglas en inglés de Word Error Rate). • Distancia de Levenshtein.

IDENTIFICADOR: RS-28
Nombre: Transcripción objetivo
Necesidad: Esencial
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-12, RU-13
Descripción: La ventana de obtención de estadísticos de la transcripción debe contener un selector de archivos para permitir la elección de un archivo de texto plano que contenga la transcripción objetivo.

IDENTIFICADOR: RS-29
Nombre: Obtención de estadísticos
Necesidad: Esencial
Prioridad: Media
Fuente: Alumno
Trazabilidad: RU-12, RU-13
Descripción: Los estadísticos se obtendrán comparando las diferencias existentes entre la transcripción obtenida y el texto objetivo textos a nivel de palabra.

IDENTIFICADOR: RS-30
Nombre: Configuración de la plataforma
Necesidad: Deseable
Prioridad: Media
Fuente: Alumno
Trazabilidad: -
Descripción: La plataforma será configurable a partir de los valores almacenados en un fichero de texto plano con pares clave-valor.

Matriz de trazabilidad RU-RS

RS/RU	RU-01	RU-02	RU-03	RU-04	RU-05	RU-06	RU-07	RU-08	RU-09	RU-10	RU-11	RU-12	RU-13	RU-14	RU-15
RS-01	X			X											
RS-02								X		X					
RS-03										X		X			
RS-04										X		X			
RS-05										X		X		X	
RS-06	X			X											
RS-07	X														
RS-08															
RS-09	X														
RS-10		X	X					X							
RS-11		X	X					X							
RS-12		X	X					X							
RS-13			X					X							
RS-14	X	X	X					X							
RS-15		X	X					X							
RS-16									X						
RS-17															X
RS-18															X
RS-19															X
RS-20										X					
RS-21											X				
RS-22					X	X	X								
RS-23					X	X									
RS-24					X	X	X								
RS-25						X									
RS-26							X								
RS-27												X	X		
RS-28												X	X		
RS-29												X	X		
RS-30															

Apéndice III: Catálogo de casos de uso

En este apéndice se desarrollan de manera formal los casos de uso presentados en la sección "4.4 Casos de uso".

Los casos de uso se catalogan mediante las siguientes propiedades:

- Identificador del caso de uso.
- Nombre.
- Descripción.
- Actores.
- Precondiciones.
- Flujo normal.
- Flujo alternativo.
- Postcondiciones.

CU-001: Seleccionar módulo

Descripción:
Permite seleccionar entre los módulos de los que disponga el usuario para un determinado tipo de proceso. Esta operación se deberá realizar cinco veces (una por cada tipo de módulo).

Actores: Usuario de la plataforma

Precondiciones: El sistema debe estar arrancado. El usuario debe disponer de los módulos adecuados presentes en una ruta accesible por el sistema.

Flujo normal:

1. El usuario pulsa en configurar en la sección de módulos de la ventana principal o en el menú herramientas > Configurar...
2. Selecciona la pestaña de la ventana de configuración adecuada para el módulo que desea.
3. Obtiene la ruta a través de la ventana emergente de selección de archivos.
4. Pulsa en Aceptar.

Flujo alternativo:

1. El usuario pulsa en configurar en la sección de módulos de la ventana principal o en el menú herramientas > Configurar...
2. Selecciona la pestaña de la ventana de configuración adecuada para el módulo que desea.
3. Obtiene la ruta a través de la ventana emergente de selección de archivos.
4. Pulsa en Aceptar.
5. El sistema indica a través de una ventana de error que el módulo escogido no es correcto.

Postcondiciones: El módulo para el tipo escogido pasa a estar seleccionado.

CU-002: Configurar un módulo manualmente

Descripción:
Permite al usuario establecer ajustes personalizados en los módulos.

Actores: Usuario de la plataforma

Precondiciones: El módulo a configurar debe estar seleccionado, para lo cual se debe ejecutar previamente el caso de uso CU-001.

Flujo normal:

1. El usuario pulsa en configurar en la sección de módulos de la ventana principal o en el menú herramientas > Configurar...
2. Selecciona la pestaña de la ventana de configuración adecuada para el módulo que desea.
3. El sistema le muestra al usuario una colección de los distintos valores que pueden ser configurados.
4. El usuario rellena los valores configurables correctamente.
5. Pulsa en Aceptar.

Flujo alternativo:

1. El usuario pulsa en configurar en la sección de módulos de la ventana principal o en el menú herramientas > Configurar...
2. Selecciona la pestaña de la ventana de configuración adecuada para el módulo que desea.
3. El sistema le muestra al usuario una colección de los distintos valores que pueden ser configurados.
4. El usuario introduce un valor incorrecto.
5. El sistema indica que el valor introducido es incorrecto.

Postcondiciones: El módulo para el tipo escogido pasa a estar configurado.

CU-003: Configurar un módulo a partir de una configuración preprogramada

Descripción:
Permite al usuario establecer ajustes personalizados en los módulos a partir de una configuración preprogramada almacenada en un archivo.

Actores: Usuario de la plataforma

Precondiciones: El módulo a configurar debe estar seleccionado, para lo cual se debe ejecutar previamente el caso de uso CU-001. Debe existir un archivo con la configuración que se desea establecer en el módulo.

Flujo normal:

1. El usuario pulsa en configurar en la sección de módulos de la ventana principal o en el menú herramientas > Configurar...
2. Selecciona la pestaña de la ventana de configuración adecuada para el módulo que desea.
3. El usuario pulsa en Abrir configuración...
4. El sistema muestra una ventana solicitando la ruta del archivo de configuración.
5. El usuario obtiene la ruta a través de la ventana de selección de archivos.
6. El usuario pulsa en Aceptar.
7. El sistema rellena los campos de los valores de configuración con los obtenidos en la lectura del archivo.
8. El usuario pulsa en Aceptar.

Flujo alternativo:

1. El usuario pulsa en configurar en la sección de módulos de la ventana principal o en el menú herramientas > Configurar...
2. Selecciona la pestaña de la ventana de configuración adecuada para el módulo que desea.
3. El usuario pulsa en Abrir configuración...
4. El sistema muestra una ventana solicitando la ruta del archivo de configuración.
5. El usuario obtiene la ruta a través de la ventana de selección de archivos.
6. El usuario pulsa en Aceptar.
7. El sistema detecta un error en el formato del archivo de configuración y lo muestra por la consola.

Postcondiciones: El módulo para el tipo escogido pasa a estar configurado.

CU-004: Seleccionar archivo de audio

Descripción:
Permite al usuario seleccionar un archivo de audio sobre el que realizar la transcripción.

Actores: Usuario de la plataforma.

Precondiciones: El usuario debe haber arrancado la plataforma.

Flujo normal:

1. El usuario pulsa el menú Archivo > Seleccionar archivo de audio…
2. El sistema le muestra una ventana donde seleccionar el archivo de audio, filtrando y mostrando únicamente los que estén soportados por el sistema.
3. El usuario elige un archivo de audio.
4. El usuario pulsa en aceptar.

Flujo alternativo:

1. El usuario pulsa el menú Archivo > Seleccionar archivo de audio…
2. El sistema le muestra una ventana donde seleccionar el archivo de audio, filtrando y mostrando únicamente los que estén soportados por el sistema.
3. El usuario elige un archivo de audio.
4. El usuario pulsa en aceptar.
5. El sistema a través de la consola informa al usuario de que el archivo escogido contiene un formato erróneo o no reconocido.

Postcondiciones: Se muestra el botón para comenzar la transcripción como habilitado. El sistema rellena los valores referidos a las propiedades de la fuente de audio.

CU-005: Realizar transcripción sobre un archivo de audio

Descripción:
Efectúa la transcripción sobre un archivo de audio previamente seleccionado.

Actores: Usuario de la plataforma.

Precondiciones: El usuario debe previamente seleccionar el archivo sobre el que desea obtener la transcripción (CU-004). Del mismo modo es necesario que se haya seleccionado un módulo (CU-001). Opcionalmente se pueden haber configurado los módulos manualmente (CU-002) o a partir de una configuración preprogramada (CU-003).

Flujo normal:

1. El usuario pulsa en el icono de *play* para iniciar la transcripción.
2. El sistema comienza la transcripción, mostrando sus resultados en la caja de texto correspondiente. El audio se reproducirá de manera sincronizada con la transcripción.
3. El sistema informa mediante una ventana que ha finalizado la transcripción.

Flujo alternativo:

Error irrecuperable

1. El usuario pulsa en el icono de *play* para iniciar la transcripción.
2. El sistema comienza la transcripción, reproduciendo el audio a la vez que muestra sus resultados en la caja de texto correspondiente.
3. Se produce un error irrecuperable en cualquiera de los módulos que detiene la transcripción y la reproducción del audio.
4. El sistema informa mediante la consola que ha sucedido un error al intentar realizar la**s** transcripciones.

Postcondiciones: Se indica en la ventana de consola lo que ha durado la transcripción. Se inhabilitan los botones de stop y pause. La ventana de resultados muestra los resultados que se han conseguido obtener con la configuración seleccionada. A partir de este punto es posible obtener los estadísticos de la transcripción (CU-006).

CU-006: Obtener estadísticos de la transcripción

Descripción:
Permite al usuario evaluar mediante valores estadísticos la transcripción realizada por el sistema (transcripción resultado) en relación a la transcripción objetivo.

Actores: Usuario de la plataforma.

Precondiciones: El usuario debe haber realizado previamente una transcripción (CU-005).

Flujo normal:

1. El usuario pulsa en el menú Herramientas > Obtener estadísticos…
2. El sistema muestra una ventana para que el usuario seleccione el archivo que contiene la transcripción objetivo.
3. El usuario selecciona un archivo válido con la transcripción objetivo.
4. El sistema muestra una ventana con los estadísticos resultado de comparar la transcripción obtenida con la objetivo.

Flujo alternativo: Ninguno.

Postcondiciones: El usuario obtiene los estadísticos para poder evaluar la transcripción que ha efectuado el sistema.

Apéndice IV: Prototipo de la interfaz gráfica

En esta sección se exponen los prototipos de la interfaz gráfica desarrollados Balsamiq, incluyendo una descripción de los elementos visuales que incluye el prototipo.

Ventana principal

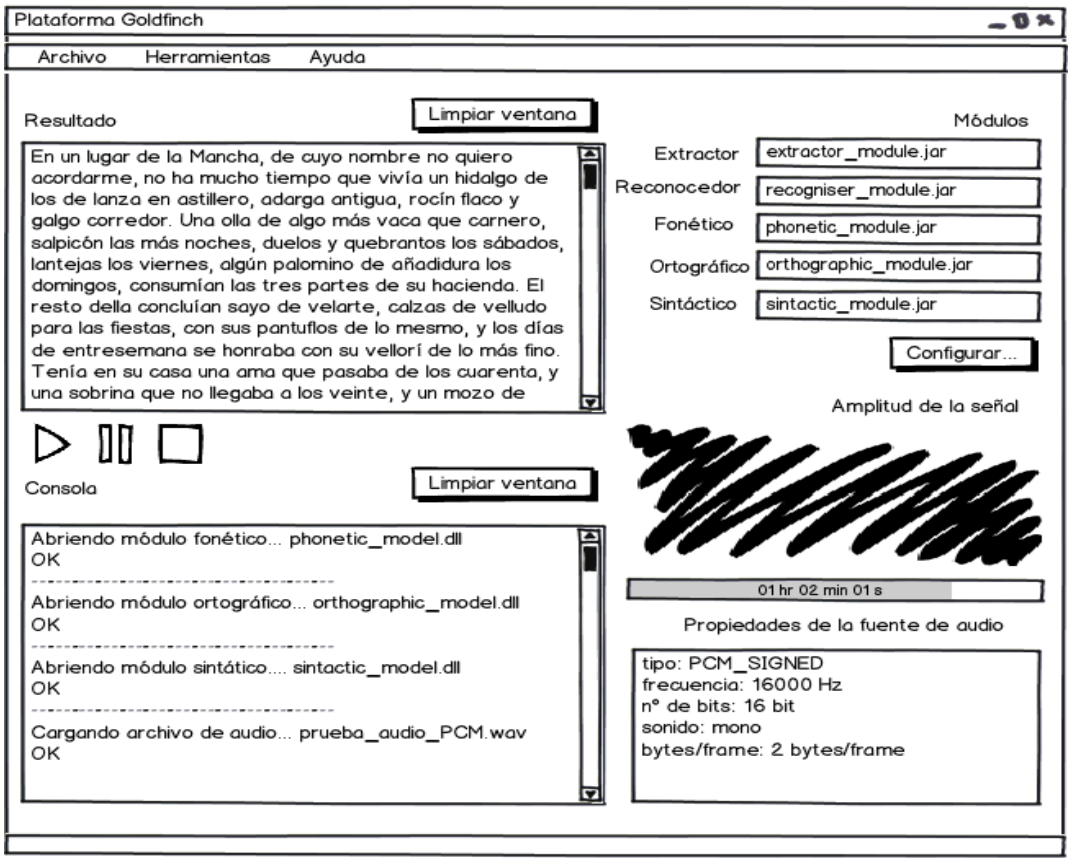


Figura 34 - Ventana principal de la plataforma Goldfinch

En la ilustración anterior se presenta la ventana principal de Goldfinch. Contiene los elementos básicos para poder operar la transcripción, modificar la configuración y selección de los módulos y obtener información.

En la caja de texto *resultado* se imprimirá el resultado de la transcripción a la vez que se ejecuta dicha operación. Dicho área de texto contiene un control para limpiar la ventana. Cada vez que se inicie una nueva transcripción esta ventana se limpiará automáticamente.

La caja de texto *consola* incluye en modo texto un histórico de los sucesos relevantes que hayan sucedido en la plataforma, como por ejemplo la

selección de módulos, la aparición de errores o la terminación de la transcripción. La funcionalidad de limpieza de la caja de texto es idéntica a la de la caja de texto *resultado*.

La caja de texto *propiedades de la fuente de audio* mostrará, un conjunto de propiedades que definen dicho archivo.

Existen tres botones para manejar el progreso de la transcripción:

- *Botón play:* Inicia la transcripción.
- *Botón pause:* Detiene la transcripción.
- *Botón stop:* Finaliza la transcripción.

La diferencia entre la detención de la transcripción y la finalización estriba en que mientras que la finalización no permite volver a reanudar la transcripción pulsando en el botón *play*, la detención si lo permite. Cuando se pulsa el botón *pause* únicamente se detendrá la transcripción momentáneamente y no se borrarán las cajas de texto de resultado y consola al reanudarse. Cuando se pulsa el botón *stop* se detendrá la transcripción y en caso de que posteriormente se pulse el botón *play*, se borrarán ambas cajas de texto.

La sección de amplitud de la señal muestra la amplitud de la señal respecto del tiempo. La barra de progreso situada debajo de la gráfica indica visualmente el porcentaje del archivo de audio que se ha procesado. La etiqueta situada en su interior muestra la misma información (en formato hh:mm:ss).

A través del botón configurar se podrá acceder a la ventana de selección y configuración de módulos, que será descrita más adelante.

Selección de módulos

En esta ilustración se muestra la ventana de selección de módulos. La funcionalidad para cada tipo de módulo se incluye en su pestaña correspondiente.

Si se selecciona el módulo correctamente no se mostrará ningún mensaje, pero si se produce un error (por ejemplo se selecciona un tipo de módulo no adecuado) se mostrará en la consola de la ventana principal.

La funcionalidad soportada por esta ventana es la descrita en el caso de uso CU-001. A continuación se muestra la ventana de selección de módulos:

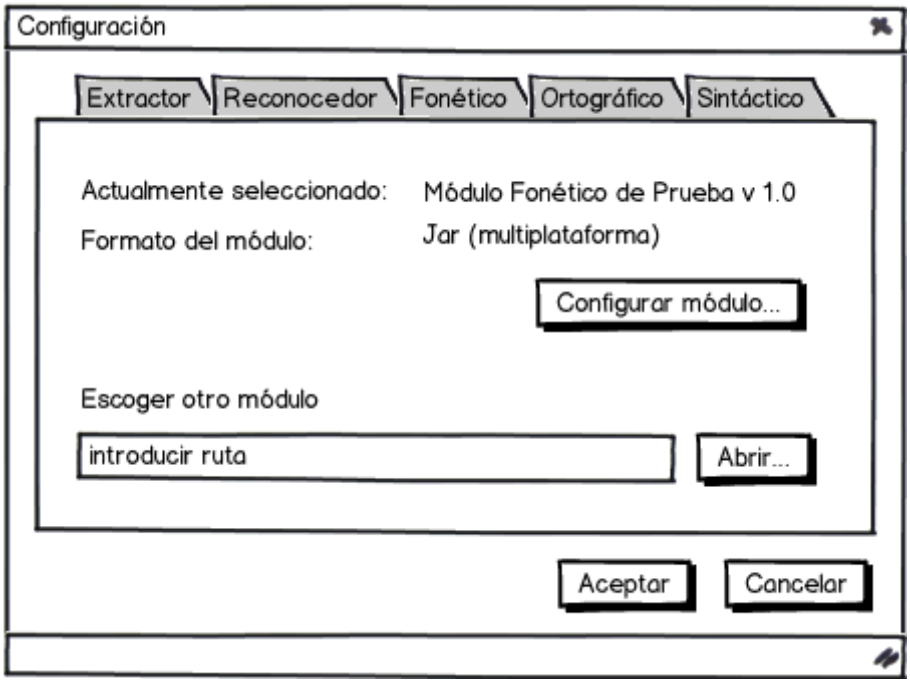


Figura 35 - Ventana de selección de módulos

Pulsando en el botón de configurar se muestra la ventana de configuración del módulo, cuya funcionalidad está descrita en la siguiente sección.

Configuración de módulos

En siguiente figura se muestra la ventana de configuración de módulos. Esta ventana podrá utilizarse con cualquier tipo módulo, y mostrará la configuración por defecto o los valores cargados de un fichero con una configuración preprogramada.

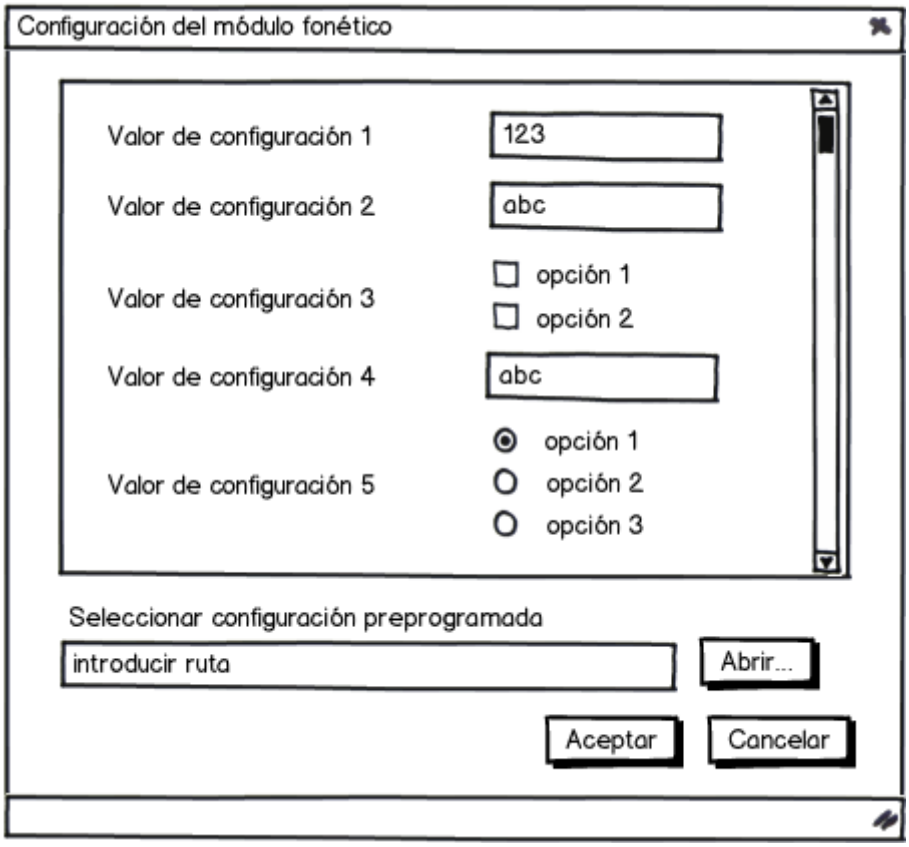


Figura 36 - Ventana de configuración de un módulo

Al abrirse por primera vez tendrá seleccionados los valores por defecto que suministre el módulo. No obstante, esta ventana permite cargar una configuración preprogramada contenida en un fichero XML.

Si existe un error de validación en la sección de valores que ha realizado el usuario, esta se mostrará en la parte superior del contenedor con barra de desplazamiento de color rojo, e indicando el mensaje de error.

El contenedor con barra de desplazamiento puede manejar los siguientes controles:

Tipo de control	Uso recomendado
Caja de texto para valores numéricos	Cuando se desea introducir un valor numérico entero. Permite restringir entre un valor máximo y un mínimo y seleccionar el valor a través de las flechas que contiene el control.
Caja de texto genérica	Cuando se desea introducir un valor numérico no entero o una cadena alfanumérica.
Checkbox	Cuando es necesario ofrecer una selección múltiple sobre un conjunto de opciones.
Radio	Cuando es necesario ofrecer una selección única sobre un conjunto de opciones.

La funcionalidad soportada por esta ventana es la descrita en los casos de uso CU-002 y CU-003.

Estadísticos de la transcripción

En esta sección se describe la interfaz gráfica que da soporte al caso de uso CU-006. A continuación se muestra el prototipo inicial de la ventana:

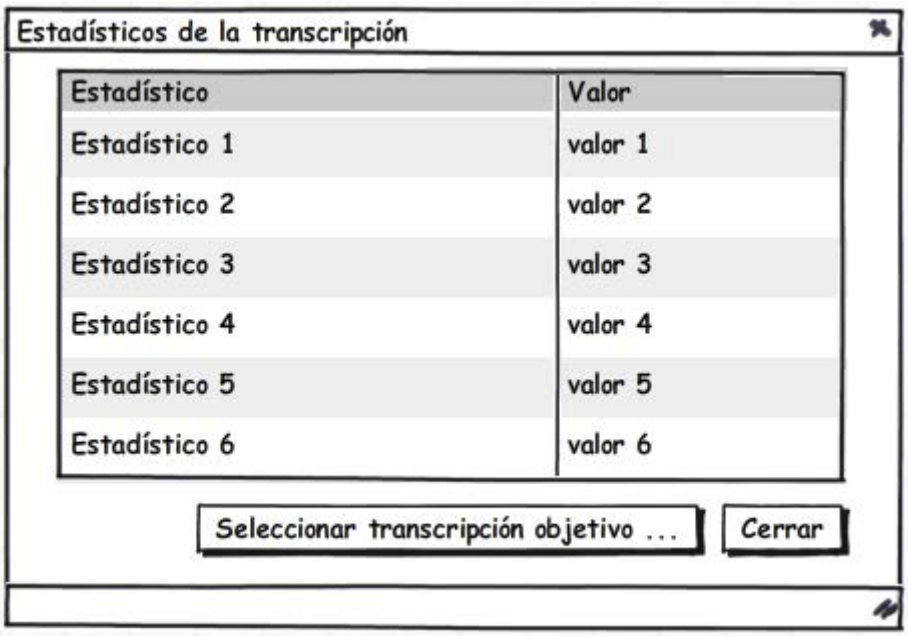


Ilustración 1 - Ventana de obtención de estadísticos de transcripción

Para que se muestren datos es necesario primero cargar la transcripción objetivo contenida en un fichero de texto plano, para lo cual es necesario pulsar en el botón "Seleccionar transcripción objetivo…". Una vez se indica donde se encuentra la transcripción objetivo, la ventana muestra los estadísticos definidos en el RS-27.

Ventana de selección de archivos

La figura siguiente muestra la ventana de selección de archivos:

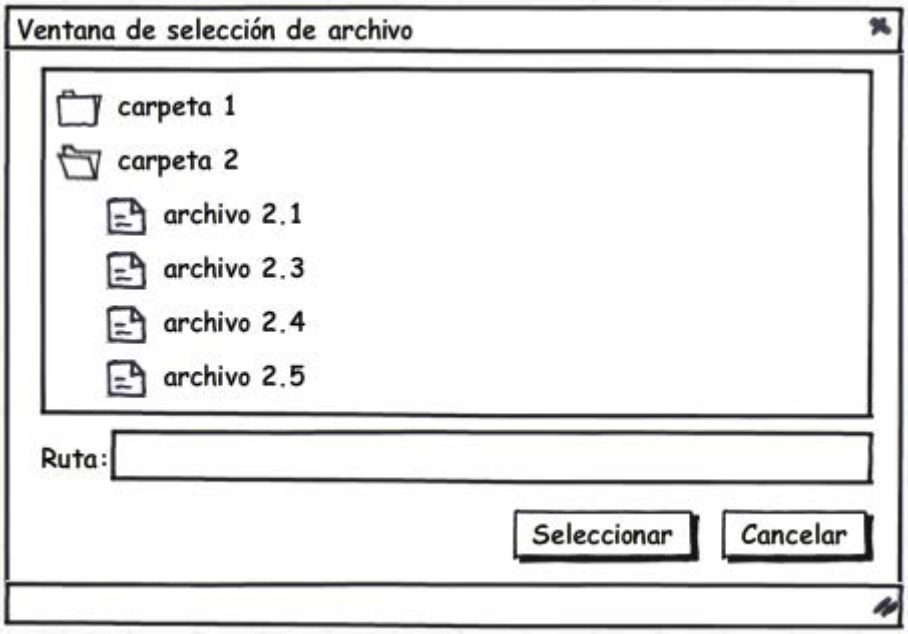


Ilustración 2 - Ventana de selección de archivos

Dicha ventana auxiliar se muestra cada vez que se necesita seleccionar un archivo. En la plataforma Goldfinch la ventana de selección de archivo se muestra en los siguientes casos:

- Cuando se desea seleccionar un módulo.
- Cuando se desea seleccionar un archivo para la transcripción.
- Cuando se debe seleccionar la transcripción objetivo para obtener los estadísticos.
- Cuando se desea escoger una configuración preprogramada.

Apéndice V: Módulos de prueba desarrollados

Al objeto de poder probar el correcto funcionamiento de la plataforma desarrollada se han creado distintos módulos de prueba que se describen a continuación.

Dichos modelos se han desarrollado a partir del pseudocódigo aportado en las últimas fases del proyecto, puesto que no estaban incluidos en el objetivo de este proyecto final de carrera.

No se incluye descripción del extractor de características, pues únicamente supuso la traducción del código de C# a Java, ni del analizador sintáctico ya que este módulo únicamente escoge la palabra de mayor probabilidad de la colección devuelta por el analizador ortográfico.

Analizador fonético

El analizador fonético utiliza un modelo basado en Gaussianas para poder realizar sus cálculos. Para ello debe cargar los valores de dicho modelo de un archivo XML al arranque de la plataforma.

El formato del fichero XML que contiene el modelo con las gaussianas se muestra a continuación:

```
<Modelo índice="1" fonema="A">
        <Gaussiana índice="1" peso ="0.3337">
                <Medias> 0.33,1.77, 25.45</Medias>
                <Covarianzas>
                        1.23,1,77,1.99,
                        1.77,2.55,0.87,
                        1.99,0.87,1.11
                </Covarianzas>
        </Gausiana>
        <Gaussiana>...</Gaussiana>
</Modelo>
```

Como se puede observar un modelo para un fonema incluye un índice, el identificador del fonema, y una colección de gaussianas. Cada gaussiana incluye un índice, un peso, una colección de medias de dimensión *n* y una colección de covarianzas de dimensión *n*x*n.*

El módulo fonético introduce el vector de características en cada uno de los modelos que conoce para obtener una colección de valores en precisión doble que representen las probabilidades de ocurrencia de un fonema.

Con cada vector de características y cada uno de los modelos se ejecuta la función para calcular el Modelo de Mezclas Gaussianas (GMM). El Modelo de Mezclas Gaussianas viene determinado por la siguiente función:

$$p(x|\lambda) = \sum_{i=1}^{M} W_i P_i(x)$$

Como se puede observar, se trata de la suma ponderada de las M densidades componentes siendo w_i el peso de cada una de ellas. Cada Gaussiana viene a su vez dada por:

$$p_i(x) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_i|^{1/2}} \quad -\frac{1}{2}(x-\mu_i)'(\Sigma_i)^{-1}(x-\mu_i)'$$

Donde μ_i es el vector de medias de dimensión *n*x1 y Σ_i es la matriz *n*x*n* de covarianzas.

Para ejecutar la función se ha utilizado una clase de Java de útiles disponible en internet.

Analizador Ortográfico

El módulo ortográfico utiliza para calcular las probabilidades de las palabras que devuelve como resultado un modelo que contiene una colección de palabras a modo de lemario. Cada palabra contenida en el modelo contiene un identificador, su fonética y su probabilidad como atributos. El formato del XML que contiene el lemario es el siguiente:

```
<Diccionario>
      <Palabra id="baba" fonetica="baba" probabilidad="72.27" />
      <Palabra id="daba" fonetica="daba" probabilidad="80.01" />
      <.../>
</Diccionario>
```

Todas las palabras contenidas en el fichero XML se cargan al arrancar el módulo.

El módulo ortográfico para cada palabra contenida en el lemario efectúa las siguientes operaciones:

Primero divide la transcripción fonética de la palabra en sus correspondientes fonemas introduciéndolo en una matriz de caracteres. Posteriormente se calcula la media de las probabilidades obtenidas del módulo anterior para cada uno de los fonemas identificados y que están contenidos en cada una de las palabras. La media obtenida multiplicada por la probabilidad leída del modelo junto con la palabra conformará cada uno de los elementos del resultado del analizador ortográfico.

Las palabras que se puedan formar parcialmente con los fonemas disponibles se marcan como tales y sufren una penalización configurable en su probabilidad (actualmente se divide su probabilidad de aparición por dos).